Bachelorarbeit zum Thema

Entwicklung einer Applikation zur Layoutoptimierung von Webseiten mit evolutionären Algorithmen und interaktiver Fitness

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

vorgelegt dem

Fachbereich Informatik und Medien der

Fachhochschule Brandenburg

Robert Fischer

18.08.2008

Erstprüfer: Dipl.-Inform. Ingo Boersch

Zweitprüfer: Prof. Dr.-Ing. Jochen Heinsohn

Inhaltsverzeichnis

1. Einführung

1.1. Aufgabenstellung

Cascading Stylesheets stellen komplexe Möglichkeiten zur Verfügung, um in HTML geschriebene Dokumente, umfangreich zu formatieren. Mit ihrer Hilfe ist es möglich die ursprünglich für HTML gedachte Aufgabe der Strukturierung von Dokumenten komplett von der visuellen Formatierung zu trennen, um wohl strukturierte Dokumente zu erstellen, die trotzdem allen optischen Ansprüchen genügen.

In den letzten Jahren wurde sehr stark der Ansatz verfolgt, dem Nutzer Dokumente im World Wide Web optisch so auszuliefern, wie sie der Designer auf seinem eigenen Rechner vorbereitet hat. Dies führt und führte zu massiven Problemen, da zum einen die unterschiedlichen Browser bestimmte Formatelemente nicht einheitlich darstellen, zum anderen ist die gewünschte Form der Darstellung nicht nur vom Autor eines Dokumentes abhängig, sondern in erster Linie von dessen Betrachtern. Es ist also wünschenswert Schnittstellen zu schaffen, die den Betrachter eines HTML-Dokumentes in die Lage versetzen Cascading Stylesheets nach seinen eigenen Anforderungen zu entwickeln.

Bestandteile der Arbeit sind eine theoretische Auseinandersetzung mit den Themen Evolutionäre Algorithmen und interaktive Fitness. Des weiteren wird versucht die Möglichkeiten beim Einsatz interaktiver Fitness im Zusammenhang mit CSS-Layouts darzustellen, außerdem wird eine prototypische GUI-Applikation vorgestellt, die eine interaktive Bewertung von CSS-Layouts realisieren soll. Weitere Bestandteile der Arbeit sind eine Implementierungsbeschreibung, so wie ein Kapitel zur Beschreibung der Anwendung und die Vorstellung einiger GUI-Konzepte zur interaktiven Bewertung.

1.2. Motivation

In den letzten Jahren hat sich das World Wide Web von einem Informationsmedium immer mehr zu einem Informations- und Unterhaltungsmedium entwickelt. War der ursprüngliche Ansatz noch die Bereitstellung und Suche nach wissenschaftlichen Informationen, so drängen heutzutage immer mehr multimediale Unterhaltungselemente in den Vordergrund. Dies schlägt sich vor allem in Aussehen und Formatierung der verschiedenen Präsenzen nieder. Immer mehr Anbieter übertreffen sich mit immer bunteren und Aufmerksamkeit fordernden Seiten und Portalen.

Der ursprüngliche Anspruch an HTML als Strukturierungssprache ging dabei immer mehr verloren, Struktur- und Formatierungsinformationen wurden vermischt und führen damit zu einem schwer wartbaren, schwer indizierbaren und unübersichtlichen Code. Ein Werkzeug um diese unübersichtlichen Strukturen wieder aufzulösen, wurde vom Word Wide Web Consortium (W3C) in Form von CSS zur Verfügung gestellt. CSS bietet die Möglichkeit eine klare Trennung von Struktur- und Formatinformationen der Webdokumente in unterschiedlichen Dateien vorzunehmen.

In jüngerer Zeit kommt auch immer mehr der Anspruch zum Tragen barrierefreie Internetpräsenzen zu erstellen um auch körperlich eingeschränkten Menschen Zugriff auf die Informationsvielfalt im World Wide Web zu bieten. Dabei ergeben sich zwei Hauptanforderungen an barrierefreie Dokumente, zum einen müssen die Dokumente wohl strukturiert sein, um auch Screenreadern oder anderen technischen Hilfsmitteln einen sinnvollen Zugriff auf die Dokumente zu erlauben. Zum anderen muss sich die Formatierung der Dokumente individuell auf den jeweiligen Betrachter anpassen lassen. Menschen mit einer Sehschwäche bevorzugen große und klare Schriftarten um ermüdungsfrei Dokumente zu lesen, Menschen mit Farbschwächen bevorzugen bestimmte Farbkontraste um die Lesbarkeit von Dokumenten zu verbessern. Oft liegt es daher nicht in der Macht des Redakteurs eine Formatierung zur Verfügung zu stellen, die für alle Betrachter der Seite gleich tauglich ist.

Die Darstellung ist vielmehr abhängig von den individuellen Bedürfnissen und Vorlieben des jeweiligen Betrachters. An dieser Stelle will die vorliegende Arbeit anknüpfen, in dem untersucht werden soll, in wie weit sich evolutionäre Techniken eignen, die Betrachter in die Lage zu versetzen Cascading Stylesheets nach ihren eigenen Bedürfnissen zu entwickeln.

2. Evolutionäre Algorithmen und interaktive Fitness

2.1. Evolutionäre Algorithmen (EA)

Evolutionäre Algorithmen sind stochastische (zufallsabhängige) Suchverfahren zur Optimierung von Problemlösungen, sie orientieren sich an unterschiedlichen Vorbildern der natürlichen Evolution und entlehnen Vorgänge und Begriffe aus der Biologie um damit Verfahren zur Lösung von Optimierungsproblemen zu beschreiben und zu realisieren.

Ein zentraler Begriff ist dabei der Begriff der Population, sie stellt die Menge aller möglichen Lösungskandidaten dar. Die einzelnen Lösungsansätze werden als Individuen bezeichnet. In einer über mehrere Generationen andauernden Suche werden einzelne Individuen an Hand ihrer Fitness ausgewählt und mit Hilfe evolutionärer Reproduktions- Operatoren neue Individuen erzeugt, diese bilden wieder neue Generationen.

Die von Generation zu Generation fortschreitende Entwicklung führt zu Individuen die immer besser der jeweiligen Ziel/Problem -stellung angepasst sind, bieten damit eine immer mehr dem Optimum angenäherte Lösung. Voraussetzung hierfür ist der Einsatz von evolutionären Operatoren die aus - im Sinne der Aufgabenstellung - guten Individuen, tatsächlich bessere Nachkommen produzieren.

2.2. Struktur Evolutionärer Algorithmen

Vor Anwendung eines EA muss eine Anfangspopulation erzeugt werden, hierbei werden für die Parameter der einzelnen Individuen in vielen Fällen Zufallswerte im zulässigen Definitionsbereich verwendet. Die Initialisierung kann aber auch problemspezifisch erfolgen. Nach Bewertung durch die Zielfunktion entsteht so die erste Generation. Bei der ebenfalls

zu den evolutionären Algorithmen zählenden genetischen Programmierung, wird die Startpopulation aus zufälligen Bäumen definierter Tiefe erzeugt.

Nach Initialisierung der Anfangspopulation startet der evolutionäre Kreislauf. Die Erstellung immer neuer und immer besser optimierter Generationen von Individuen wird so lange fortgesetzt bis das definierte Abbruchkriterium erfüllt ist. Abbruchkriterien können je nach Problemstellung z.B. ein bestimmter Zielfunktionswert, eine maximale Anzahl von Generationen oder eine bestimmte Zeitspanne sein.

Mit Hilfe des Zielfunktionswertes wird so lange der EA arbeitet, jedem Individuum der Population im Vergleich zu allen anderen Individuen eine Fitness zugewiesen. Zum besseren Verständnis zeigen die beiden folgenden Abbildungen eine mögliche Population von Farbkontrasten und eine mögliche Population von Schriftarten.

Abbildung 1: Beispiel - Population von Farbkombinationen (aus [Le00])

Abbildung 2: Beispiel - Population von Schriftarten (aus [Le00])

2.3. Fitness Allgemein

Der Fitnesswert beschreibt die Wahrscheinlichkeit, dass ein Individuum, ganz oder teilweise, mit in die nächste Generation eingeht. Dies wird auch als positive Selektion bezeichnet und impliziert gleichzeitig eine negative Selektion, für den Fall das Individuen auf Grund ihres Fitnesswertes nicht selektiert werden.

„Entsprechend dieser Fitness werden die Individuen (Eltern) für die Produktion von neuen Individuen (Kindern) ausgewählt. (*Selektion*).“ [Po00]

Grundsätzlich können drei Arten der Fitness(-funktion) unterschieden werden: Berechenbare Fitness, interaktive Fitness und implizite Fitness.

Die berechenbare Fitness ist die am häufigsten auftretende Variante, es lässt sich eine Zielfunktion aufstellen und die Fitness kann an Hand des errechneten Zielfunktionswertes zugewiesen werden. Der Zielfunktionswert hängt dabei nur von den Variablen eines Individuums ab, der Fitnesswert dagegen ist relativ im Vergleich zu allen anderen Individuen des Pools zu sehen.

Die implizite Fitness stellt sich nicht als skalare Größe dar, sie ist anders als die berechenbare Fitness keine absolute Größe. Vielmehr ist sie ähnlich wie in der Natur, als Fitness im System zu verstehen und beschreibt die Überlebensfähigkeit eines Individuums in einem begrenzten Lebensraum. Also seine Fähigkeit sich in Konkurrenz zu allen anderen Individuen der Population zu behaupten.

Bei der interaktiven Fitness übernimmt der Mensch, an Stelle einer Zielfunktion, die Bewertung der einzelnen Individuen subjektiv mit Hilfe einer Mensch/Maschine Schnittstelle.

2.4. Fitnesszuweisung

Übliche Arten der Fitnesszuweisung sind die proportionale Fitnesszuweisung (*proportional fitness assignment*), die reihenfolgenbasierte Fitnesszuweisung (*rank-based fitness assignment*) und bei Kombination mehrerer relevanter Zielfunktionswerte, die Mehrkriterielle Fitnesszuweisung.

2.4.1. Proportionale Fitnesszuweisung

Bei der proportionalen Fitnesszuweisung werden den einzelnen Individuen Fitnesswerte zugewiesen, die proportional zu ihren errechneten Zielfunktionswert sind. Die wichtigsten Skalierungen sind lineare Skalierung, linear dynamische Skalierung, logarithmische Skalierung und exponentielle Skalierung.

Beim Einsatz der proportionalen Fitnesszuweisung treten zwei entscheidende Probleme auf, zum einen werden die guten Individuen einer Generation sehr stark bevorzugt, zum anderen ist eine signifikante Unterscheidung der guten Lösungen an Hand der Fitness kaum mehr möglich.

2.4.2. Reihenfolgenbasierte Fitnesszuweisung

Die bei der proportionalen Fitnesszuweisung auftretenden Probleme werden bei der reihenfolgenbasierten Fitnesszuweisung minimiert. Hierzu werden alle Individuen, abhängig von ihren errechneten Zielfunktionswerten, in eine sortierte Liste eingefügt. Die Fitness der einzelnen Individuen bestimmt sich aus deren Position innerhalb der Liste.

Pohlheim unterscheidet lineares und nicht- lineares Ranking um den Selektionsdruck bei der reihenfolgenbasierten Fitnesszuweisung zu variieren. [Po00]

2.4.3. Mehrkriterielle Fitnesszuweisung (Mehrzieloptimierung)

Im Unterschied zur proportionalen und der reihenfolgenbasierten Fitnesszuweisung, bei denen jeweils nur ein Zielfunktionswert entscheidend ist, werden bei der mehrkriteriellen Fitnesszuweisung zur Gütebestimmung eines Lösungskandidaten mehrere Kriterien betrachtet. Erst durch den Vergleich aller lösungsrelevanten Kriterien kann in vielen Fällen entschieden werden, welches Individuum, im Sinne der Aufgabe, besser als ein anderes ist. Aus den Vergleichen kann, ähnlich wie bei der einkriteriellen Fitnesszuweisung, eine Reihenfolge aufgestellt werden. (Mehrkriterielles Ranking) Auf dieser Ordnung lassen sich die oben genannten einkriteriellen Methoden anwenden.

Bei der mehrkriteriellen Betrachtung ist oft nicht entscheidbar welches Individuum „besser“ als ein anderes ist. Sich widersprechende Lösungsanforderungen und/oder das Fehlen einer Parameter-Kombination, die eine optimale Lösung aller einzelnen Anforderungen garantiert, führen zu Problemen bei der Entscheidbarkeit.

Zur Lösung dieser Probleme können unterschiedliche Ansätze verfolgt werden. Die zufällige Auswahl der anzuwendenden Fitnessfunktionen betrachtet von Generation zu Generation nur eine zufällige Anforderung an die Lösungskandidaten. Eine andere Möglichkeit ist die Wichtung der einzelnen Fitnessfunktionen. Die Fitness eines Individuums wird hierbei durch die Summe aller seiner gewichteten Fitnessfunktionswerte bestimmt. Dabei erlauben es Wichtungsfaktoren, den unterschiedlichen Lösungsanforderungen, eine relative Wichtigkeit zuzuordnen, um so Lösungen im Hinblick auf ein bestimmtes Kriterium zu optimieren.

Ein weiterer Ansatz ist die Ordnung der Population nach Dominanz der einzelnen Individuen, dies wird auch als Pareto-Optimierung bezeichnet. Es wird davon ausgegangen, dass ein Individuum ein anderes dominiert, wenn es in allen Einzel- Fitnesswerten mindestens

genauso gut ist, in einem Fitnesswert aber besser. Schrittweise wird dann allen nicht dominierten Individuen der Rang 1 zugeordnet, anschließend werden sie aus der Population entfernt, nun wird wieder allen nicht dominierten Individuen der Rang 2 zugeordnet und sie werden ebenfalls entfernt, dieser Vorgang wird, so lange gewünscht, fortgesetzt. Eine Wichtung der einzelnen Fitnessfunktionen ist dabei, anders als bei der Arbeit mit Wichtungsfaktoren, nicht notwendig.

2.5. Selektion

Die Selektion ist der eigentliche Prozess der Auswahl von Individuen an Hand ihrer Fitness, die selektierten Individuen dienen als Eltern für die Erzeugung von Nachkommen. Diese Nachkommen werden so lange in eine neue Population eingefügt, bis diese aufgefüllt ist, anschließend wird die alte Generation verworfen. Etwas anders verhält es sich bei der Ersetzungsselektion (steady-state), hier verdrängen die Nachkommen direkt andere Individuen der gleichen Population. Die Wahrscheinlichkeit eines Individuums selektiert zu werden ergibt sich aus dem Fitnesswert des Lösungskandidaten und der eingesetzten Selektionsmethode.

Der Selektionsdruck beschreibt die Wahrscheinlichkeit, dass, verglichen mit der durchschnittlichen Selektionswahrscheinlichkeit, die tatsächlich besten Individuen einer Population ausgewählt werden. Der Verlust an Vielfalt (loss of diversity) beschreibt dabei die Menge der Individuen des Selektionspools, die während der Selektion nicht ausgewählt werden. „Individuen, ... die bei einer vollständigen Ersetzung der Elternpopulation durch Nachkommenpopulationen verloren gehen würden.“ [Po00] Die Selektionsintensität beschreibt den erwarteten durchschnittlichen Fitnesswert aller Individuen nach einer durchgeführten Selektion.

Folgende wichtige Selektionsvarianten sind zu unterscheiden.

2.5.1. Rouletteselektion

Bei der Rouletteselektion wird der gesamte Selektionspool als Linie abgebildet. Proportional zur Fitness der einzelnen Individuen werden den Individuen verschieden große Abschnitte auf dieser Linie zugeteilt. Eine gleichverteilte Zufallszahl, im Wertebereich von 0 bis zur Summe aller Fitnesswerte, bestimmt welches Individuum ausgewählt wird. Der Vorgang wird so lange wie nötig fortgesetzt.

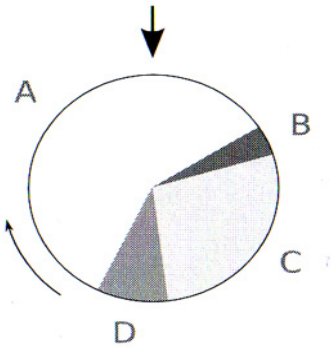

Abbildung 3: Rouletteselektion auf einer Population mit 4 Individuen (aus [BHS07])

Die Rouletteselektion birgt eine Reihe von Problemen. Das Zentralste ist, dass der Zufall den Selektionsprozess so weit überdeckt, dass die Verteilung der Nachfolgegeneration oft nicht der Fitnessverteilung der Elterngeneration entspricht. Nicht unwahrscheinlich ist ebenfalls die Möglichkeit, dass auf Grund des Zufalls, das schlechteste Individuum oder in einem Generationszyklus nur schlechte Individuen ausgewählt werden.

2.5.2. Stochastic Universal Sampling

Das Stochastic Universal Sampling ist eine Modifikation der Rouletteselektion, die das beschriebene Hauptproblem umgeht. Dazu werden zur Auswahl von n- Individuen, n- äquidistante Punkte (Zeiger) auf der Linie angeordnet. Nur der Startpunkt ist zufällig gewählt, jeder Zeiger bestimmt ein selektiertes Individuum.

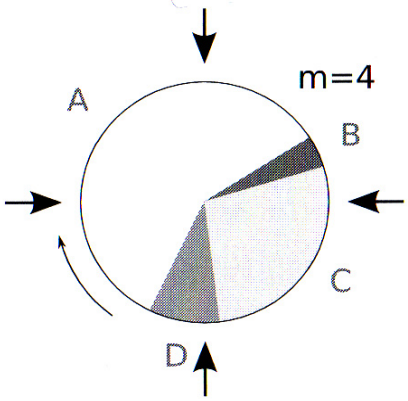

Abbildung 4: Stochastic Universal Sampling - Population mit 4 Individuen (aus [BHS07])

2.5.3. Turnierselektion

Bei der rangbasierten Turnierselektion werden aus der Population zufällig n- Turniergruppen gebildet. Sieger eines Turniers ist das Individuum mit der höchsten Fitness innerhalb einer Turniergruppe. Der Selektionsdruck kann über die Größe der Turniergruppen variiert werden, von 1 (ein Individuum pro Turniergruppe) - kein Selektionsdruck - bis zur maximalen Anzahl aller Individuen in einer Turniergruppe - maximaler Selektionsdruck. Es werden so viele Turniere ausgetragen, wie Individuen auszuwählen sind.

2.5.4. Truncation Selection

Die Truncation Selection (Abschneide Selektion) ist eine rangbasierte Selektion und wählt jeweils nur die besten Individuen einer Population aus, hierzu werden die Individuen entsprechend ihrer Fitness sortiert. Mit Hilfe eines Schwellwertes werden alle Individuen deren Fitnesswert größer als der Schwellwert ist ausgewählt, alle deren Fitnesswert kleiner ist werden verworfen. Es existieren nur die Selektionswahrscheinlichkeiten 0 oder 1.

In der Praxis bietet sich eine Kombination mit der Elite Strategie an, bei der das beste Individuum stets in die nächste Generation übernommen wird. Der Selektionsdruck lässt sich beim Stochastic Universal Sampling über die Anzahl der Zeiger, bei der Turnierselektion über die Größe der Turniergruppen und bei der Abschneide Selektion über den Schwellwert steuern.

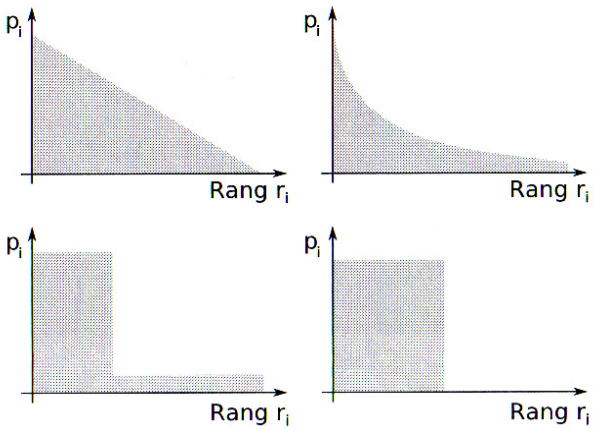

Abbildung 5: Rangbasierte Selektionsverfahren (Linear-,Exponentiell-, Greedy-, Abschneideselektion) (aus [BHS07])

Die folgende Abbildung zeigt alle vorgestellten Selektionsverfahren mit ihren spezifischen Eigenschaften, sortiert nach steigendem Selektionsdruck, zusammengefasst in einer Tabelle.

Selektions-verfahren	Rouletteselektion	Stochastic Universal Sampling	Turnierselektion	Truncation Selektion
Anzahl der ausgewählten Individuen je Durchgang	- ein Individuum je Durchgang	- Menge der gewählten Zeiger = Menge der selektierten Individuen	- ein Individuum je Durchgang	- abhängig von der Größe der Population und des gewählten Schwellwertes
Art der Zuweisung	- fitnessproportional	- fitnessproportional	- rangbasiert	- rangbasiert
Voraussetzung	- diskrete Fitnesswerte	- diskrete Fitnesswerte	- rangbasierte Ordnung der Individuen	- rangbasierte Ordnung der Individuen
Selektionsdruck	- nicht variierbar	- variierbar durch die Anzahl der Zeiger	- variierbar durch die Anzahl / Größe der Turniergruppen - diskriminierendes Verfahren, die n schlechtesten Individuen haben keine Chance zur Reproduktion	- variierbar durch den Schwellwert - diskriminierendes Verfahren, die n schlechtesten Individuen haben keine Chance zur Reproduktion
Verlust an Vielfalt	- relativ niedrig da zufällige Auswahl - jedes Individuum hat eine Chance (evtl. sehr gering) seine Eigenschaften weiterzugeben	- relativ niedrig da zufällige Auswahl - jedes Individuum hat eine Chance (evtl. sehr gering) seine Eigenschaften weiterzugeben	- abhängig der der Größe der Turniergruppen - wenige Turniergruppen = großer Verlust an Vielfalt	- abhängig vom Schwellwert - allg. etwas höher als bei SuS oder TS, da die schlechten Individuen überhaupt keine Chance bekommen ihre Eigenschaften weiterzugeben
Vorteile	- einfache Anwendung	- Selektionsdruck variierbar - minimiert die Probleme der Rouletteselektion - „Für die fitness-proportionale Selektion ist bis heute keine besseres (schnelleres) Verfahren gefunden worden [Po00 S.26]	- Selektionsdruck variierbar - auch anwendbar wenn keine diskreten Fitnesswerte vorliegen	- Selektionsdruck variierbar - auch anwendbar wenn keine diskreten Fitnesswerte vorliegen - schnelle Einengung des Suchbereiches auf vielversprechende Lösungen
Nachteile	- Verteilung der Folgegenration spiegelt in der Regel kaum die Fitnessverteilung der Vorgängergeneration wieder - diskrete Fitnesswerte benötigt	- diskrete Fitnesswerte benötigt		- relativ hoher Verlust an Vielfalt durch das Abschneiden der schlechten Individuen

Abbildung 6: Übersicht der Selektionsverfahren - geordnet nach Selektionsdruck

2.6. Interaktive Fitness

Im Gegensatz zur bisher betrachteten Fitnessbewertung, die auf berechenbaren Zielfunktionen fußte, basiert die interaktive Fitness auf der subjektiven Bewertung durch menschliche Individuen. Es können drei verschiedene Räume unterschieden werden, in denen sich die einzelnen Individuen einer Population befinden. Der Parameterraum repräsentiert den Suchraum, in ihm befinden sich die Genotypen der Individuen, also die Codierungen der einzelnen Lösungskandidaten mit ihren individuellen Parameterwerten. Der Darstellungsraum enthält die Phänotypen der Individuen, also die konkrete Ausprägung der Individuen abhängig von ihrem Genmaterial (Codierung). Die Lage der Individuen im Bewertungsraum, der bei interaktiver Fitness auch als psychologischer Raum bezeichnet wird, ergibt sich aus der Bewertung durch den Nutzer.

Die Grundidee bei der interaktiven Optimierung ist, dass Individuen die in einem der drei Räume benachbart oder dicht beieinander liegen, auch in den anderen beiden Räumen eng beieinander liegen. Oft gilt, dass ähnliche Genotypen ähnliche Phänotypen hervorbringen und ähnliche Phänotypen auch eine ähnliche Fitness haben, also ähnlich bewertet werden.

Wichtige Kriterien zum sinnvollen Einsatz interaktiver Fitness sind, die schwierige Berechenbarkeit oder Nichtberechenbarkeit der Zielfunktion, oder deren Teile und die Möglichkeit einer für den Menschen anschaulichen und leicht fassbaren Darstellung der einzelnen Lösungskandidaten. Interaktive Fitness kann bspw. sinnvoll eingesetzt werden, wenn es um bestimmte Vorlieben bei Audio-, Video- oder Grafikmaterial geht. Hier werden die besten Ergebnisse durch eine menschliche Bewertung, basierend auf subjektiven Eindrücken, Vorlieben, Emotionen und Empfindungen, erzielt.

Interaktive Fitness ist eine Optimierungsmethode die auf der menschlichen Bewertung basiert, alle anderen Eigenschaften evolutionärer Algorithmen werden unverändert genutzt. Dabei werden nach Takagi zwei Hauptdefinitionen unterschieden. Eine etwas engere Definition: „Die Technik der interaktiven Fitness optimiert ein Zielsystem, basierend auf subjektiven menschlichen Bewertungen als Fitnessparameter für das Ergebnis.“ [Ta01] Hierbei kann der Mensch durch die Bewertung einzelner Individuen oder deren Parameter Einfluss auf den evolutionären Prozess nehmen.

Die etwas weitere Definition lautet: „Die Technik der interaktiven Fitness optimiert ein Ziel-

system mit Hilfe einer interaktiven Mensch-Maschine Schnittstelle.“ [Ta01] Die etwas weitere Definition erlaubt es den Benutzer in eine Rolle zu versetzen, die über die reine Bewertung hinausgeht. Der Mensch kann eher in den Rang eines Steuermanns erhoben werden, der aktiv auf den EA Prozess Einfluss nimmt, indem er z.B. direkt die Selektionswahrscheinlichkeit eines einzelnen Individuums bestimmt oder von vornherein ganze Regionen einer Population als unbrauchbar verwirft.

2.6.1. Praxisbeispiel – EvoFIT

Als konkretes praktisches Beispiel soll hier das Programm „EvoFIT“ genannt werden. Der vollständige Name des Programms lautet „Evolving the face of a criminal“, was frei übersetzt werden könnte: „Das Gesicht eines Kriminellen entwickeln“. Aufgabe des Programms ist die Erstellung von Phantombildern mit Hilfe evolutionärer und interaktiver Techniken. Dabei werden dem Nutzer in endlich vielen Durchgängen sechs Bilder von Gesichtern präsentiert, der Nutzer vergibt für die einzelnen Darstellungen Fitnesspunkte. Mit fortschreitender Anzahl von Generationen nähert sich das Bild immer mehr dem konkreten Bild der gesuchten Person an.

Bisher arbeiteten ähnliche Verfahren immer nur mit einem Bild und der Nutzer musste fortlaufend die Unterschiede zu dem gesuchten Bild beschreiben. Dies stellt eine schwierige Aufgabe dar, da der Nutzer sich ständig die Unterschiede zwischen dem aktuell dargestellten Bild und dem Bild in seiner Erinnerung vor Augen führen und diese auch artikulieren muss. Ein Gesicht wieder zuerkennen ist dagegen eine deutlich weniger ermüdende Aufgabe. In unterschiedlichen Testreihen mit zwei Personengruppen, wurden den Teilnehmern verschiedene Gesichter vorgelegt, die sie zwei Tage später zur Erstellung eines Phantombildes beschreiben sollten.

Dabei hat sich gezeigt, dass die Personengruppe, welche die Gesichter aus der Erinnerung beschreiben sollte, deutlich schlechter abschnitt, als die Gruppe die mit Hilfe des Programms „EvoFIT“ das Fahndungsbild interaktiv entwickelte. Das Programm wird inzwischen konkret für die Erstellung von Fahndungsbildern eingesetzt und ist kommerziell verfügbar. [EV08]

Die folgende Abbildung zeigt in vier Einzelbildern beispielhaft, von links nach rechts, die Ergebnisse von vier Generationszyklen. Das rechte vierte Bild ist das Endergebnis des Prozesses.

Abbildung 7: Beispiel für die schrittweise Erarbeitung von Phantombildern mit EvoFIT (aus [EV08])

2.6.2. Fitnesszuweisung und Selektionsvarianten bei interaktiver Fitness

Der Einsatz möglicher Fitnesszuweisungen und Selektionsvarianten im Zusammenhang mit interaktiver Fitness, hängt sehr stark vom Design der Mensch/Maschine Schnittstelle und der Anzahl und Granularität (Auflösung) der vom Nutzer erwarteten Eingaben ab. Grundsätzlich können absolute und relative Fitness unterschieden werden.

2.6.3. Absolute Fitness

Bei der Betrachtung von absoluter Fitness können im wesentlichen drei verschiedene Auflösungen unterschieden werden. Der Mensch übernimmt die bewertende Funktion und abhängig von seiner Wertung wird den Individuen eine Fitness zugewiesen.

Das *„Schulklassenmodell“* ist ein punktebasiertes Wertungssystem, bei dem der Benutzer die einzelnen Individuen, ähnlich wie Schüler einer Klasse, mit Noten/Punkten von 1 bis n bewertet. Bei diesem Modell sind grundsätzlich alle vorgestellten Fitnesszuweisungen und Selektionsverfahren anwendbar. Je feiner die Bewertungsskala aufgelöst wird und je mehr Wertungen für den Nutzer möglich sind, um so bessere und schnellere EA Ergebnisse sind zu erwarten. Allerdings vergrößert dies auch die Langeweile des Nutzers und führt zu schnellerer Ermüdung. Takagi bezeichnet diesen Effekt als „quantisation noise“. [Ta01]

Im *„Drei Klassen Modell“* wird die Anzahl der möglichen Wertungen eingeschränkt, dem Nutzer stehen nur noch drei Optionen zur Verfügung. Beispielsweise könnte eine Einteilung „Good“, „Neutral“ und „Bad“ gewählt werden. Der Nutzer bewertet die Individuen nur mit diesen Eingabemöglichkeiten und ordnet die Individuen so in drei Klassen ein.

Eine mehrkriterielle oder proportionale Fitnesszuweisung ist hier nicht anwendbar, da keine aussagekräftigen Kriterien vorhanden sind, bzw. eine signifikante Unterscheidung der Individuen an Hand ihrer Fitness nicht mehr möglich wäre. Aus den entstandenen drei Gruppen lassen sich die Individuen nur noch zufällig auswählen. Dies könnte z.B. zu der Selektion einer großen Anzahl zufälliger „Good“, einer mittleren Anzahl zufälliger „Neutral“ und jeweils nur eines zufälligen „Bad“ Individuums führen. Auf Grundlage der drei entstandenen Fitnessgruppen wäre auch die Abschneide Selektion anwendbar indem man nur die „Good“ oder nur die „Good“ und „Neutral“ Individuen auswählt.

Das *„mehrkriterielle Modell“* erfordert eine so weit aufgelöste Schnittstelle, dass der Nutzer einzelne Kriterien der Lösung individuell bewerten kann. Bei der Bewertung einer Grafik könnten z.B. Auflösung, Helligkeit, Kontrast oder ähnliche Eigenschaften getrennt bewertet werden. Eine andere Möglichkeit wäre es mehrere Personen in den Evaluierungsprozess einzubeziehen und sie jeweils nur einen Aspekt der Lösung bewerten zu lassen. Bei diesem Modell lassen sich ähnlich wie beim „Schulklassenmodell“, je nach Auflösung der Wertungsskala für die einzelnen Kriterien, alle vorgestellten Selektionsverfahren und Fitnesszuweisungen anwenden.

2.6.4. Relative Fitness

Anders als bei der absoluten Fitness gibt es bei der relativen Fitness keinen konkreten Fitnesswert. Statt die Parameter eines einzelnen Lösungskandidaten zu bewerten, wird das Individuum im Vergleich zu allen anderen Individuen des Pools betrachtet und die Frage aufgeworfen, welches Individuum besser „gefällt“. Dies wird auch als implizite Fitness oder Fitness im System bezeichnet.

Der Zwischenschritt der eigentlichen Fitnessbestimmung, wie er in den vorigen Abschnitten erläutert wurde, kann dabei entfallen. Der Mensch übernimmt nun nicht mehr nur eine bewertende Funktion, sondern er kann als Steuermann den EA Prozess aktiv beeinflussen. Zum Beispiel indem er Individuen direkt selektiert oder die Selektionswahrscheinlichkeit einzelner Individuen beeinflusst.

Für solche Bewertungsansätze werden innovative GUI-Schnittstellen benötigt. Denkbar wären Systeme, in denen ähnliche Individuen zusammen angeordnet auf dem Bildschirm dargestellt werden, der Benutzer erhält nun die Möglichkeit seine „Gunst“ zu verteilen. Er könnte ähnlich einem Sonnenschein, besonders interessante Regionen der Population zum „Wachsen“ anregen. In den stärker beschienenen Regionen steigt die Selektions-

wahrscheinlichkeit an, in den weniger stark beschienenen Regionen nimmt sie entsprechend ab. Obwohl in einem solchen Modell keine direkte Fitnesszuweisung statt findet, könnte aus dem Alter der Individuen eine Fitness abgeleitet werden. Die ältesten Individuen wurden durch den Nutzer lange „gepflegt“ und sind vermutlich sehr wertvoll, dass Alter könnte indirekt als Fitnesswert betrachtet werden.

2.6.5. Probleme beim Einsatz interaktiver Fitness

Der Einsatz interaktiver Fitness führt in der Praxis zu einer Reihe von Problemen. Das größte Problem liegt dabei in der Mensch/Maschine Schnittstelle selbst und der nachlassenden menschlichen Ausdauer und Konzentration. Die menschliche Ermüdung bei der Bewertung von Lösungskandidaten an einem ermüdungsfreien Rechner führt dazu, dass der Evaluierungsprozess nach einer begrenzten Anzahl von Generationen nicht fortgesetzt werden kann. Dies kann den Einsatz von interaktiver Fitness in der Praxis einschränken.

Als direkte Folge dieses Problems treten weitere Probleme auf. Es stellt sich erstens die Frage: wie findet man eine optimale Lösung in einer sehr kleinen Population und mit wenigen Generationen? Die evolutionären Algorithmen müssen unter schwierigen Bedingungen angewendet werden. Die Anzahl der gleichzeitig auf einem Anzeigegerät darstellbaren Lösungskandidaten ist stark begrenzt, die menschliche Aufnahmefähigkeit für gleichzeitig dargestellte Individuen ist eingeschränkt, und die menschliche Ermüdung während des Evaluierungsprozesses muss minimiert werden.

Das dritte Problem stellen zeitabhängige Bewertungen durch den Nutzer dar. Sollen z.B. Filme oder Sounds evaluiert werden, so muss er die aktuell dargestellten Mediendaten aus dem Gedächtnis mit vorigen Darstellungen vergleichen und bewerten. Dies schränkt die Aufnahmefähigkeit zusätzlich ein und führt zu schnellerer Ermüdung.

Letztlich bewertet ein Mensch während eines Evaluierungsprozesses auch nicht nach unveränderlichen Maßstäben, vielmehr entwickelt sich seine Vorstellung von einer optimalen Lösung im Laufe des Prozesses. Diese „Veränderung der Fitnessfunktion“ führt im Besonderen bei der Fitnessprognose zu Schwierigkeiten auf die im Folgenden noch näher eingegangen werden muss. Bei der Implementierung interaktiver Fitness muss sichergestellt werden, die aus den aufgeführten Einschränkungen folgenden Probleme zu minimieren. Nur so können verwertbare Ergebnisse erzielt und die kostbare menschliche Bewertungsleistung optimal genutzt werden.

2.6.6. Lösungsansätze

Glücklicherweise sind viele durch Evolutionäre Algorithmen zu bearbeitende Problemfelder, nicht zwingend auf eine große Anzahl von Generationen angewiesen um zufriedenstellende Resultate zu erzielen. Dies macht eine Anwendung auch unter den vorher genannten Einschränkungen durchaus sinnvoll und führt zu guten Lösungen.

Da populationsbasierte Algorithmen den Suchraum an vielen verschiedenen Punkten untersuchen, bringen sie zufällig oft schon in der Grundpopulation einige dem Optimum angenäherte Lösungen hervor. Diese Eigenschaft und die Möglichkeit gleichzeitig mehrere Bereiche des Suchraumes darzustellen, führen zu einer für den Nutzer weniger ermüdenden Bewertung, als dies beispielsweise bei gradientenbasierten Verfahren der Fall wäre.

Auf subjektiven Faktoren beruhende Bewertungen unterscheiden sich grundlegend von mathematischen Optimierungen. Bei der menschlichen Evaluierung genügt es oft einen optimalen Bereich für die Lösung zu erreichen, es ist nicht erforderlich, dass exakte Optimum zu bestimmen. Aus diesem Grund ist es möglich auch mit wenigen Generationen ein zufriedenstellendes Ergebnis zu erzielen.

Das Hauptproblem bei interaktiver Fitness bleibt die Ermüdung des Menschen, diesem Problem muss bei der Implementierung besonderes Augenmerk gelten.

2.6.6.1 Methoden zur Eingabe mit wenigen diskreten Fitness- Werten

Die menschliche Ermüdung bei der Bewertung einzelner Lösungskandidaten ist stark von der Art abhängig wie die Werte eingegeben werden. Bei einer großen Auswahl an Eingabemöglichkeiten fällt es uns schwerer zwischen den einzelnen Wertungen zu unterscheiden und die individuellen Lösungskandidaten effektiv zu evaluieren. Dies hat zusätzliche Ermüdung zur Folge.

Als Beispiel soll hier die Einschätzung der Temperatur aufgeführt werden. Es fällt uns sicher sehr schwer zwischen konkreten Werten wie z.B. 26°C und 27°C zu unterscheiden. Wird aber die Auswahl der Eingabemöglichkeiten auf die drei Auswahlmöglichkeiten „kalt“, „neutral“ und „heiß“ eingeschränkt, fällt es uns schon deutlich leichter eine Bewertung vorzunehmen. Wenige Eingabemöglichkeiten führen zu einer signifikanten Verminderung der menschlichen Ermüdung während der Evaluierung und damit zu besseren Ergebnissen der Bewertung. Nach Takagi führt eine zu große Auswahl von Eingabemöglichkeiten zu einem „Quantisierungsrauschen“, dass die Ergebnisse des EA verschlechtern kann. [Ta01]

2.6.6.2 Prognose der Fitnessbewertung

Um das zweite beschriebene Problem, die relativ kleinen Populationen und geringe Anzahl von Generationen, zu minimieren, können Methoden eingesetzt werden, die versuchen die Fitness vorauszusagen. Sollte es möglich sein eine Voraussage für die Fitness zu treffen, so kann die Leistung des EA durch den Einsatz sehr großer Populationen erhöht werden. Um den Nutzer nicht zu ermüden, werden ihm aus diesen großen Populationen nur wenige Lösungskandidaten, die eine gute Fitness- Voraussage haben, zur Evaluierung präsentiert.

Eine Form der vorausschauenden Fitnessbewertung ist der Einsatz einer abstandsbasierten Funktion. Die Fitnessvoraussage ergibt sich hierbei aus dem euklidischen Abstand der einzelnen Individuen, zu den vom Nutzer favorisierten Individuen. Grundlage hierfür ist, dass jedes Individuum in einen konkreten Parametervektor überführt werden kann, mit dessen Hilfe sich die Abstandsberechnung realisieren lässt.

Eine weitere abstandsbasierte Voraussage ist der Einsatz regelbasierter Systeme. Die Idee ist, dem Nutzer die Möglichkeit zu bieten sehr gute Lösungskandidaten auszuwählen, den nicht gewählten Individuen werden dann, auf Basis eines Regelsystems, „bias“ Fitnesswerte anstelle einer Fitness von 0 zugewiesen. Im einfachsten Fall, kann jedem nicht gewählten Individuum die gleiche „bias“ Fitness zugewiesen werden. Wie im vorigen Absatz beschrieben, ist es auch hier möglich die Fitness auf Grundlage des Euklidischen Abstandes zu bestimmen.

Als problematisch erweisen sich bei der vorausschauenden Fitnessbetrachtung vor allem zwei Punkte. Voraussagen die auf der euklidischen Distanz basieren, berücksichtigen nicht die evtl. unterschiedliche „Wertigkeit“ der einzelnen Lösungsparameter. Dieses Problem, lässt sich aber über eine Wichtung der einzelnen Fitnessfunktionswerte und die Einbeziehung von Wichtungsfaktoren, wie in Kapitel 2.4.3 beschrieben, lösen bzw. minimieren.

Weiterhin ist die subjektive Evaluierung durch menschliche Nutzer nicht absolut, sondern muss relativ zur aktuell dargestellten Generation betrachtet werden. Die Meinung des Nutzers entwickelt und ändert sich im Verlauf des Evaluierungsprozesses. Es ist möglich, dass weiter zurückliegende Generationen anders wahrgenommen wurden als die aktuelle Generation. Die daraus folgende unterschiedliche Wertung, macht es mitunter schwierig die Nutzermeinung vorherzusagen.

2.6.6.3 Motivierende Interfacegestaltung

Ein zentrales Problem beim Einsatz interaktiver Fitness stellt die Nutzerschnittstelle dar. Je feiner und genauer die Bewertung des Nutzers aufgelöst wird, desto besser werden die EA Ergebnisse. Fein granulierte Eingabemöglichkeiten stehen aber einem ermüdungsfreien Bewertungsprozess, wie in den vorigen Kapiteln beschrieben, direkt entgegen.

Der Evaluierungsprozess sollte so abwechslungsreich wie möglich gestaltet werden, verschiedene Versuche haben ergeben, dass die besten Ergebnisse erzielt werden, wenn der Bewertungsprozess „fesselnd“ fast schon „spielerisch“ umgesetzt wird. Die Eingabemöglichkeiten sollten so weit wie möglich eingeschränkt werden. [Ta01]

Eine deutliche Erleichterung für den Nutzer stellt sich ein, wenn während des gesamten Bewertungsprozesses ein „Elite Individuum“ dargestellt wird. Dies versetzt den Nutzer in die Lage die aktuelle Bewertung ständig mit dem bisher erreichten Optimum abzugleichen. Verschiedene Untersuchungen haben gezeigt, dass diese Form der Schnittstelle signifikante Verbesserungen bei der Bewertung zur Folge hat. [Ta01]

Die Untersuchung verschiedener grafischer Oberflächen hat ebenfalls ergeben, dass der Bewertungsprozess deutlich effektiver ist, je weniger Interaktion dem Nutzer abgefordert wird. So wurde in verschiedenen Tests festgestellt, dass schon ein zusätzlicher Mausklick beim Bewertungsprozess zu deutlich schlechteren Ergebnissen führen kann. [Ta01]

Der Gestaltung der grafischen Oberfläche muss also besondere Aufmerksamkeit gelten, Ziel ist es zwischen Ermüdung und Auflösung der Eingaben einen Mittelweg der minimalen Ermüdung und maximalen EA Leistung zu erreichen.

2.6.6.4 Parallele Bewertung

Ein anderer Ansatz die Ermüdung des Benutzers während des Evaluierungsprozesses zu minimieren, ist die parallele Bewertung. Hierbei werden einzelne Individuen hinsichtlich ihrer Güte, nicht nur von einem sondern von mehreren Nutzern parallel bewertet. Am Beispiel CSS könnte dies bedeuten, dass ein Nutzer nur die Farbgebung, ein anderer nur die Anordnung und ein weiter Nutzer nur die Schriftgestaltung bewertet. Dies führt für den einzelnen Nutzer zu einem deutlich weniger komplexen Bewertungsprozess, ergibt aber in der Summe aller Bewertungen eine sehr genau aufgelöste Beurteilung.

Problematisch bei dieser Form der Bewertung ist, dass verschiedene Menschen bestimmte Aspekte wie Farbkontraste nicht homogen bewerten. Voraussetzung für diese Bewer-

tungsart ist eine Gruppe von Personen, die ähnliche Lösungsparameter auch ähnlich bewerten. Andernfalls würde der gewonnene Vorteil bei der parallelen Bewertung, durch ein uneinheitliches Gesamtergebnis wieder negiert werden.

2.6.6.5 Archivierung aller erreichten Lösungen

Ein weiterer Ansatz ist die Archivierung aller erreichten Lösungsergebnisse mit ihren jeweiligen Wertungen. Der Nutzer hat nun die Möglichkeit seine Bewertung zeitunabhängig vorzunehmen. Da er nicht mehr gezwungen ist aktuelle Generationen aus dem Gedächtnis heraus mit vorangegangen zu vergleichen. Viel mehr hat er die Möglichkeit, sich alle erreichten Generationsschritte noch einmal anzeigen zu lassen und so einen deutlich besseren Vergleich, auch über einen längeren Zeitraum, zu ziehen. Voraussetzung dafür ist, dass sich der Evaluierungsprozess entsprechend pausieren lässt und der Nutzer die Möglichkeit bekommt, auf die archivierten Ergebnisse zuzugreifen.

3. HTML und CSS

3.1. Geschichte und Entstehung HTML und CSS

In den frühen Jahren des Internet, war HTML noch eine sehr schlanke Sprache die nur aus Elementen zur Strukturierung von Dokumenten bestand. So konnten beispielsweise Elemente wie Absätze, Hyperlinks, Listen oder Überschriften beschrieben werden. Ursprünglich war die Sprache dabei als reine Beschreibungssprache für verschiedenen Bestandteile von Dokumenten gedacht, auf das eigentliche Erscheinungsbild eines Dokumentes hatte HTML dabei keinen oder nur sehr geringen Einfluss. Dies impliziert jedoch auch, dass es keine Möglichkeit gab sicher zu stellen, dass ein Dokument beim Leser so dargestellt wird, wie es der Autor geplant hat.

Mit dem Aufblühen des WWW und der Entstehung von immer neuen und immer mehr Webseiten wurde durch die Autoren der Ruf nach Möglichkeiten laut, auch die Präsentation der einzelnen Elemente zu steuern. Ursprünglich war dies für HTML nicht vorgesehen, doch unter dem entstandenen Druck flossen erste Elemente zur Steuerung der Präsentation mit in die Sprache ein. Die ersten Markup Elemente dienten dabei der Darstellung von z.B. fett gedrucktem oder kursiven Text, plötzlich diente die Sprache nicht mehr nur der strukturellen Beschreibung von Dokumenten, sondern enthielt gleichzeitig Elemente um die Präsentation der Inhalte zu beschreiben.

3.2. Vermischung von Struktur- und Formatelementen

Was in den nächsten Jahren folgte war ein großes Durcheinander, immer neue Elemente zur Beschreibung der Präsentation flossen, teilweise ohne definierte Konventionen, in die Sprache ein. Die Folge war eine immer stärkere Vermischung von Struktur- und Formatan-

weisungen. Einzelne Browseranbieter fügten zu der ohnehin schon unübersichtlichen Anzahl von Elementen noch proprietäre Tags hinzu um beispielsweise Text zum Blinken oder zum Rotieren zu bringen. Der ursprüngliche Anspruch zur Strukturierung der Elemente eines Dokumentes ging dabei mehr und mehr verloren, strukturell betrachtet waren und sind viele Seiten plötzlich nicht mehr als eine zufällige Aneinanderreihung von Zeichenketten.

Als Beispiel soll hier auf das font-Tag verwiesen werden, dieses erlaubt die Schrift-Formatierung einzelner Textabschnitte. Das font-Tag hat allerdings keinerlei Bedeutung für die Struktur eines Dokumentes. Erstellt und formatiert nun ein Autor seine Überschriften mit Hilfe des font-Tags und nicht mit Hilfe des vorgesehenen Heading Tags, bleiben keinerlei Strukturinformationen vorhanden. Ein Browser oder Screenreader ist nicht mehr in der Lage diese Textzeilen als Überschriften zu erfassen und von anderen Textzeilen zu unterscheiden, lediglich die Formatierung zeichnet sie für das menschliche Auge noch als solche aus. Der Hauptgrund für die Vermischung von Struktur- und Formatanweisungen war der Anspruch vieler Autoren, die Seiten dem Leser so zu präsentieren, wie sie selbst sie designt haben. Doch diese Vermischung führt zu einer Vielzahl schwerwiegender Probleme.

3.3. Probleme unstrukturierter Dokumente

Das Hauptproblem unstrukturierter Seiten stellt die schwierige Indizierung durch Suchmaschinen oder ähnliche automatisierte Agenten dar. In einem gut strukturierten Dokument kann eine leistungsstarke Suchmaschine beispielsweise die Suche in bestimmten Kapiteln oder Abschnitten realisieren und bietet so eine deutliche Zeitersparnis bei der Suche innerhalb bestimmter Abschnitte. Durch eine wohl geformte Struktur der Dokumente lässt sich sowohl eine bessere Indizierung der einzelnen Kapitel, wie auch eine bessere Position innerhalb der Suchmaschinen gewährleisten, da viele Suchmaschinenbetreiber die Struktur eines Dokumentes mit zur Bestimmung seiner Güte heranziehen.

Ein weiterer wichtiger Nachteil ist die verminderte Zugänglichkeit von unstrukturierten Dokumenten, so können Screenreader ohne das Vorhandensein von Strukturinformationen beispielsweise nur das gesamte Dokument und nicht einzelne Abschnitte vorlesen. Eine wohlgeformte Struktur jedoch würde es dem Screenreader erlauben z.B. nur Überschriften vorzulesen, der Nutzer hat dann die Möglichkeit zu entscheiden was ihn wirklich interessiert und in welche Bereiche er tiefer eindringen möchte.

Letztlich führt eine saubere Struktur auch zu leichter wartbaren Dokumenten, die sich mit weniger Aufwand anpassen und verändern lassen. Bei einer gut aufgebauten Internetpräsenz ist es ausreichend die Formatierungsparameter an einer Stelle anzupassen, um alle betreffenden Ausgaben mit einem Arbeitsschritt zu verändern.

Der Anspruch geht also dahin, die Struktur eines Dokumentes mit einer ansprechenden Seitendarstellung zu verbinden, hier bieten Cascading Stylesheets (CSS) die Lösung. Bereits im Jahr 1996 wurden CSS vom World Wide Web Consortium (W3C) uneingeschränkt empfohlen und erhielten damit das gleiche Gewicht wie HTML selbst.

3.4. CSS als Lösung der Probleme

CSS bietet sehr viel umfangreichere Möglichkeiten der Gestaltung eines Dokumentes, als dies in HTML jemals möglich war. Für jedes beliebige Element können mit CSS Text- und Hintergrundfarben, Rahmen, Außen- und Innenabstände, Textformatierungen und viele weitere individuelle Formatierungen vorgenommen werden.

Ein wesentlicher Vorteil dabei ist, dass die Befehle für die visuelle Darstellung nicht mehr über das gesamte Dokument verstreut sind, sondern Struktur- und Darstellungsanweisungen sauber getrennt und in zentralen Dateien abgelegt werden können. Anstatt für jedes Element die Formatierung redundant im HTML-Dokument zu hinterlegen werden diese zusammengefasst in einer CSS-Datei abgelegt. Bei der Änderung der visuellen Darstellung ist es nicht mehr nötig, dies an vielen einzelnen Stellen kompliziert im HTML-Code vorzunehmen, vielmehr ist eine Änderung im CSS oftmals schon ausreichend um die Anzeige aller betreffenden Elemente anzupassen. Dadurch ist es möglich die Darstellung des gesamten Dokumentes von einer einzelnen zentralen Stelle aus zu steuern und anzupassen.

Eine weitere mächtige Eigenschaft von CSS ist die Kaskadierung, sie ermöglicht es globale Darstellungsregeln zu definieren, die dann bei Bedarf überschrieben werden können. Grundsätzlich gilt dabei die Regel, dass ein Element, dass keine eigenen Stilinformationen besitzt die Stilinformationen des Elternelementes bzw. umschließenden Elementes erbt. Ein einfaches Beispiel zur Verdeutlichung dieser Arbeitsweise ist folgendes.

Es ist möglich für das Body-Tag eines HTML Dokumentes, also das absolute Oberelement, Informationen zur visuellen Darstellung zu definieren, alle weiteren Elemente die sich innerhalb des Bodys eines HTML-Dokumentes befinden erben dann diese Informationen und werden entsprechend dargestellt. Sollten für einzelne Bereiche andere Darstel-

lungen gewünscht sein, beispielsweise eine andere Schriftart, so lässt sich dies nur für dieses eine spezifische Element überschreiben, ohne die restliche Darstellung zu berühren.

Neben der einfachen und zentralen Änderbarkeit der Stilinformationen bietet CSS jedoch auch Vorteile die vielleicht nicht auf den ersten Blick erkennbar sind. Da Informationen zur Darstellung nicht mehr an mehreren Stellen redundant im HTML Code vorgehalten werden müssen, führt dies zu deutlich kleineren Dateigrößen, damit zu weniger genutzter Bandbreite bei der Übertragung der Dokumente. Die Seiten können dem Leser somit schneller, effektiver und kostengünstiger ausgeliefert werden. Letztlich soll hier noch auf den Vorteil der personalisierten Darstellung eingegangen werden.

Durch das Einbinden verschiedener CSS ist es möglich ein Dokument in vielen verschiedenen Arten darzustellen. Abhängig von den subjektiven Vorlieben des Nutzers, kann dieser sich beispielsweise für ein bestimmtes Farbschema oder eine bestimmte Schriftart entscheiden, welche ihm persönlich am besten gefällt. Dies bietet dem Nutzer mehr individuellen Komfort bei der Darstellung der Dokumente.

Doch abseits von persönlichen Vorlieben und Komfort bietet diese Option weitreichende Möglichkeiten zur barrierefreien Darstellung von Dokumenten. So ist es ohne Probleme möglich dem Nutzer CSS-Vorlagen mit verschiedenen Schriftgrößen anzubieten. Menschen mit einer Sehbehinderung erhalten so die Möglichkeit die Darstellung der Seite ihren eigenen Bedürfnissen anzupassen. Nicht wenige Menschen haben Probleme bei der Darstellung bestimmter Farbkombinationen, die Rot-Grün-Schwäche ist dabei sicher eines der bekanntesten Wahrnehmungsprobleme. Selbstverständlich kann es nicht das Ziel sein, solche Leser schon durch die Formatierung eines Dokumentes auszuschließen. Deshalb ist es wünschenswert verschiedene, auf bestimmte Wahrnehmungsschwächen abgestimmte Formatinformationen, zu hinterlegen und dem Leser die Möglichkeit zu geben die für ihn angenehmste Form auszuwählen.

3.5. Motivation für interaktiven Ansatz

Die letzten Abschnitte haben deutlich gemacht, dass es nicht vom Autor einer Seite abhängig sein sollte wie die Dokumente dem Leser präsentiert werden, viel mehr sind es die persönlichen Vorlieben oder eventuellen Einschränkungen des Lesers die die Darstellung

bestimmen sollten. Dies führt uns zu den in Kapitel 1 beschriebenen Möglichkeiten von interaktiver Fitness. Wäre es nicht die allerbeste Lösung wenn dem Leser die Möglichkeit gegeben wird, die Darstellung des Dokumentes nach seinen eigenen Vorstellungen zu entwickeln?

An diesem Punkt wird die vorliegende Arbeit anknüpfen, indem untersucht werden soll, in wie weit es realisierbar ist dem Nutzer, mit den Möglichkeiten von Evolutionären Algorithmen und interaktiver Fitness, eine individuelle Entwicklung von CSS zu ermöglichen.

4. Anwendungskonzeption

4.1. Zielbestimmung

Das zu entwickelnde Programm EvoCSS soll den Benutzer in die Lage versetzen, mit Hilfe einer grafischen Oberfläche, Cascading Stylesheets zu entwickeln, die seinen persönlichen Vorlieben bzw. Anforderungen entsprechen. Dabei kommen zur Bewertung der einzelnen Designvorschläge interaktive Elemente zum Einsatz, die Optimierung der einzelnen Stylesheets wird mit Hilfe eines evolutionären Verfahrens realisiert.

Die Anwendung soll dabei sehr einfach aufgebaut sein, so dass auch Nutzer ohne spezielle Kenntnis der Techniken HTML und CSS das Programm problemlos bedienen können.

4.1.1. Musskriterien

4.1.1.1 Der Benutzer

- der Benutzer kann mit Hilfe einer einfachen grafischen Oberfläche verschiedene Stylesheets bewerten, die immer wieder mit dem selben HTML Gerüst dargestellt werden
- der Benutzer hat die Möglichkeit den Bewertungsprozess jederzeit neu zu beginnen
- der Benutzer hat die Möglichkeit den einzelnen CSS Darstellungen Fitnesspunkte zu geben und so direkt den Fortbestand einzelner Lösungen zu beeinflussen
- der Benutzer hat die Möglichkeit nach Bewertung den nächsten Generationsschritt anzustoßen
- die Mutations- und Crossoverwahrscheinlichkeit können während des Prozesses verändert werden

- die Wichtungsfaktoren können im Programmverlauf geändert werden

4.1.1.2 Der evolutionäre Kreislauf

- das Programm kann eine endliche Menge von unterschiedlichen Stylesheets (Individuen) vorhalten, von denen neun gleichzeitig dem Benutzer zur Bewertung angezeigt werden
- zum Startzeitpunkt des Programms wird eine Population von Individuen mit zufälligen Parametern innerhalb der erlaubten Wertebereiche erzeugt
- jedes CSS soll dabei entsprechend codiert und durch einen Parametervektor repräsentiert werden
- mit Hilfe genetischer Operatoren sollen die einzelnen Stylesheets, im Sinne der Nutzerbewertung, immer weiter optimiert werden, um zu einem für den Nutzer angenehmen Endergebnis zu gelangen
- im Zuge der Elitestrategie wird das best-bewertete Individuum mit in die nächste Generation übernommen
- als genetische Operatoren sollen Crossover und Mutation angewendet werden, die Mutation soll dabei eine zu starke Verarmung an bestimmten Eigenschaften innerhalb des Populationspools vorbeugen

4.1.1.3 Die Oberfläche

- die Oberfläche besteht aus drei zentralen Elementen, einer Menüleiste, einer Steuerbox im linken Bereich und einer Präsentationsbox im rechten Bereich der Oberfläche
- die Steuerbox bietet Zugriff auf die mit der Evolution verknüpften Funktionen des Programms, hier kann der Kreislauf neu gestartet werden, die nächste Generation erzeugt und weitere Funktionen aufgerufen werden
- die Präsentationsbox stellt über den gesamten Programmverlauf neun mögliche Lösungskandidaten dar, weist deren momentane Anzahl von Fitnesspunkten aus und gibt dem Nutzer die Möglichkeit einzelne Individuen zu bewerten

4.1.1.4 Das System

- das Programm soll in deutscher Sprache erstellt werden

- die in der Präsentationsbox zur Anzeige kommenden Stylesheets werden als Kombination einer HTML- und einer CSS-Datei temporär gespeichert
- die erzeugten Dateien können direkt oder als Vorlage für HTML+CSS-Layouts eingesetzt werden
- die Struktur muss so offen programmiert werden, dass sich die HTML Struktur jederzeit auch komplex erweitern lässt um umfangreichere Layouts zu repräsentieren
- die grafische Oberfläche soll modular aufgebaut sein, um leicht Änderungen und Erweiterungen vornehmen zu können

4.1.2. Wunschkriterien

- beliebige HTML+CSS-Layouts lassen sich über eine Schnittstelle mit dem Programm verarbeiten
- das Programm kann mehrsprachig genutzt werden
- der Evolutionskreislauf lässt sich abspeichern und zu einem späteren Zeitpunkt fortsetzen
- die Bewertung erfolgt nicht mehr statisch und sequentiell sondern mit Hilfe eines alternativen GUI Konzeptes, Beispiele dafür finden sich im Kapitel über ungewöhnliche GUI Konzepte
- eine Menüleiste bietet zentralen Zugriff auf alle Funktionen
- die bisher erreichten Lösungen werden gespeichert und können wieder abgerufen werden

4.1.3. Abgrenzungskriterien

- mit dem Programm können keine HTML+CSS-Layouts im eigentlichen Sinne „erstellt“ werden
- die Erweiterung des Layouts und das Einfügen von Inhaltselementen ist nur im Programmcode möglich, dies soll aber einfach und entsprechend dokumentiert sein
- es kann immer nur ein jeweiliger Evolutionskreislauf gleichzeitig realisiert werden
- das Programm ist nicht netzwerkfähig

4.2. Produkteinsatz

4.2.1. Anwendungsbereich

Benutzer ohne HTML- oder CSS-Kenntnisse, sollen durch dieses Programm in die Lage versetzt werden, abhängig von ihren persönlichen Vorlieben bzw. Wahrnehmungsschwächen, ein für sie optimales Stylesheet zu entwickeln, dass sowohl ihrem ästhetischen Empfinden, wie auch körperlichen Voraussetzungen genügt.

4.2.2. Zielgruppen

- Auftraggeber für Internetpräsenzen, die so in die Lage versetzt werden interaktiv bestimmte Darstellungen der Präsenz zu bewerten.
- Betreiber von Internetpräsenzen die über die Einbindung dieser Funktionalität ihren Besuchern die Möglichkeit geben, die Darstellung der Seite auf deren persönliche Vorlieben / Anforderungen abzustimmen.
- So lange keine weiteren Sprachen integriert sind, muss der Benutzer die Programmsprache deutsch zumindest verstehen.

4.3. Produktumgebung

4.3.1. Software

Das Programm benötigt für den Betrieb einen PC mit Microsoft Windows, aber Version Windows XP SP“, weiterhin werden der Internet Explorer und die dot.net Laufzeitumgebung, mindestens in der Version 2, benötigt.

4.3.2. Hardware

Das Programm läuft auf jedem PC mit grafischer Benutzeroberfläche, der auch die Anforderungen von Microsoft Windows XP erfüllt.

4.4. Produktfunktionen

/PF000/

Das Programm erzeugt mit zufälligen Parameterwerten innerhalb des jeweiligen Wertebereichs eine Ausgangspopulation von n-Individuen. Von dieser Gesamtpopulation werden neun Individuen dem Benutzer zur Bewertung dargestellt.

/PF010/

Der Benutzer kann den Evolutionsvorgang neu starten, dies entspricht dem Neustart des Programms. Es werden alle Einstellungen auf den Ausgangszustand zurückgesetzt und eine neue zufällige Ausgangsgeneration erzeugt.

/PF020/

Eine Auswahl der einzelnen Entwürfe kann dargestellt und bewertet werden, der Benutzer erhält die Möglichkeit über entsprechende Schaltflächen jede Darstellung individuell zu bewerten. Es können Fitnesspunkte verteilt aber auch wieder entzogen werden.

/PF030/

Über ein entsprechendes Steuerelement kann der Nutzer nach Abschluss seiner Bewertung die nächste Generation erzeugen und anzeigen lassen.

/PF040/

Die Formatierungsanweisungen sind ausschließlich in der CSS-Datei enthalten, in der HTML-Datei werden lediglich die Strukturinformationen gespeichert.

/PF050/

Die Mutations- und Crossoverwahrscheinlichkeiten sollen sich während des Programmverlaufs anpassen lassen.

/PF060/

Die Wichtungsfaktoren für Schriftart, Schriftfarbe und Farbschema sollen sich im Programmverlauf anpassen lassen.

/PF070/

Für die einzelnen Individuen sind Funktionen zu implementieren die zum Einen die unäre Mutation eines Individuums und zum anderen die binäre Operation Crossover entspre-

chend umsetzen.

/PF080/

Es ist ein Selektionsverfahren zu implementieren.

/PF090/

Es ist ein entsprechender evolutionärer Kreislauf zu implementieren, der mit Hilfe der vorgenannten Funktionen von Selektion, Mutation und Crossover eine Nachkommengeneration erzeugt.

4.5. Produktdaten

/PD000/

Jedes Stylesheet besteht aus n-Divisionen, jede Division enthält n-CSS-Attribute zur Formatierung des Containers, zu jeder Division werden die entsprechenden Attribute wie Größe, Ausrichtung, Farbe, Schriftart, Schriftgröße und -farbe gespeichert.

/PD010/

Die anzuzeigenden Layouts werden als Kombination einer HTML- und einer CSS-Datei, in Form temporärer Dateien, physikalisch auf dem Dateisystem abgelegt.

5. Implementierung der Anwendung

In den folgenden Kapiteln soll die Implementierung und das Design der erstellten Anwendung erläutert werden. Zuerst wird kurz dargestellt wie die definierten Pflichtkriterien Eingang in die Anwendung fanden. Die ausführliche Beschreibung der Umsetzung aus Entwicklersicht schließt sich dann in den weiteren Kapiteln an. Die in Kapitel 4 dargestellten Musskriterien und Programmfunktionen konnten vollständig umgesetzt werden.

5.1. Konkrete Umsetzung der Produktfunktionen und -daten

/PF000/ - Erzeugung einer zufälligen Startpopulation

/PF010/ - Reinitialisierung / Zurücksetzen des Programms durch den Nutzer

Die Applikation erzeugt nach dem ersten Start eine vollständige Population von Individuen, mit zufälligen Parameterwerten, im zulässigen Definitionsbereich. Während des Bewertungsprozesses hat der Nutzer jederzeit die Möglichkeit über die Schaltfläche „Neu“ das Programm in den Startzustand zurückzusetzen. Es wird dabei eine neue zufällige Startgeneration erzeugt, gleichzeitig werden die Parameterwerte, wie Mutations- und Selektionswahrscheinlichkeit, auf ihre Ausgangswerte zurückgesetzt.

/PF020/ - Bewertung der einzelnen Stylesheets

Über zwei Schaltflächen hat der Nutzer die Möglichkeit den einzelnen Individuen eine individuelle Fitness zuzuordnen. Ein Button dient dabei dem Inkrementieren, der andere dem Dekrementieren des aktuellen Fitnesswertes. Der Fitnesswert wird zusätzlich unter den angezeigten Individuen ausgegeben. Standardfitnesswert für alle noch nicht bewerteten Individuen ist eins.

/PF030/ - Erzeugung der nächsten Generation

Mit Hilfe der Schaltfläche „Nächste“ ist der Nutzer jederzeit in der Lage die nächste Generation von Individuen zu erzeugen. Die hierfür zugrunde liegenden Funktionen und Implementierungen werden in den kommenden Kapiteln ausführlich beschrieben.

/PF040/ - Trennung von Format- und Strukturierungsanweisungen

Die durch das Programm erzeugten HTML-Dateien entsprechen dem XHTML 1.0 Standard. Die HTML-Dateien sind komplett frei von Formatanweisungen und enthalten lediglich Informationen zu Struktur und Inhalt des Dokumentes. Die Formatanweisungen werden in einer getrennten CSS-Datei abgelegt, diese wird anschließend in die entsprechende HTML-Datei eingebunden.

/PF050/ - Mutations- und Crossoverwahrscheinlichkeit zur Laufzeit einstellen

Über zwei Schieberegler hat der Nutzer die Möglichkeit die Parameter für Mutations- und Crossoverwahrscheinlichkeit im Programmverlauf einzustellen. Die Änderungen wirken sich nach dem Drücken des „Nächste“ Buttons direkt auf die nächste Generation aus und werden als Aktualparameter für den weiteren Programmverlauf hinterlegt, so lange der Nutzer sie nicht erneut anpasst.

/PF060/ - Wichtungsfaktoren zur Laufzeit einstellen

Über drei entsprechende Schieberegler wird der Nutzer in die Lage versetzt die Wichtungsfaktoren, für Farbschema, Schriftart so wie -größe, zu konfigurieren. Die Änderungen wirken sich mit dem Drücken des Buttons „Nächste“ aus und werden bis zu einer weiteren Änderung als Aktualparameter des Programms gespeichert.

/PF070/ - Funktionen für Mutation und Crossover

Die Funktionen zur Mutation von einzelnen Individuen, so wie Crossover von zwei Individuen wurden entsprechend implementiert. Eine genauere Beschreibung zu Umsetzung und Funktionsweise der Operatoren kann den folgenden Kapiteln entnommen werden.

/PF080/ - Implementierung Selektionsverfahren

Als Selektionsverfahren wurden die Roulette Selektion und das Stochastic Universal Sampling implementiert. Eine Änderung des Selektionsverfahren ist derzeit nur direkt im Programmcode möglich. Eine genauere Beschreibung der Umsetzung beider Verfahren folgt in den nächsten Kapiteln.

/PF090/ - Evolutionärer Kreislauf

Der evolutionäre Kreislauf zur Auswahl von Individuen aus der Elterngeneration und Erzeugung einer Nachkommengeneration, mit Hilfe von Selektion, Mutation und Crossover, wurde erfolgreich implementiert.

/PD000/ - Strukturierung der Stylesheets in Divisionen und Werte

Die Stylesheets werden aus einzelnen CSS-Division Elementen zusammengesetzt, für jedes Division Element lassen sich Formatanweisungen speichern.

/PD010/ - Temporärer Ordner und temporäre Dateien

Zum Startzeitpunkt wird durch das Programm ein temporärer Ordner im lokalen Systemordner des ausführenden Nutzers erstellt. Gleichzeitig wird für die Vorschau die entsprechende Anzahl von HTML- und CSS-Dateien angelegt. Mit jedem neuen Generationsschritt werden diese Dateien entsprechend von den neuen Vorschaudateien der nächsten Generation überschrieben. Das temporäre Verzeichnis wird beim ordnungsgemäßen Beenden des Programms entfernt.

5.2. Einführung - Implementierung aus Entwicklersicht

Diese Einführung soll lediglich einen Überblick über die Implementierung der Funktionalitäten geben, eine genaue Betrachtung zur Umsetzung wird in den nächsten Abschnitten erfolgen.

Für die Implementierung der Anwendung kam die Programmiersprache C# und die Entwicklungsumgebung MS Visual Studio in der Version 2008 zum Einsatz. Ziel bei der Umsetzung war ein weitestgehend modularer Aufbau, um schnell und unkompliziert Änderungen/Erweiterungen vorzunehmen. So sind alle einzelnen Pakete der Anwendung als eigenständige Windowsbibliotheken umgesetzt worden. Auf dieser Grundlage lassen sich problemlos z.B. neue Views implementieren und auf die Anwendung aufsetzen.

Im Laufe der Implementierung kamen unterschiedliche Analyse- und Entwurfsmuster zum Einsatz. Häufig eingesetzt wurden z.B. die Analysemuster der abstrakten und konkreten Oberklassen. In der Bibliothek EvoCSS-Values werden alle möglichen CSS-Werttypen von der abstrakten Oberklasse *CssValue* abgeleitet, um sicherzustellen, dass alle benötigten Schnittstellen in den abgeleiteten Klassen implementiert werden. Ebenfalls zum Einsatz kam das Konzept von Interfaces, z.B. in der Bibliothek EvoCSS-Elements.

Als wichtigstes Entwurfsmuster wurde das Beobachtermuster implementiert. Die benötigten Operationen werden dabei durch die konkrete Oberklasse *Observable* und das Interface *ObserverIF* zur Verfügung gestellt. Beobachter Klassen müssen lediglich das Interface implementieren und die Methode *ReceiveObserverUpdate()* realisieren. Diese Methode wird von den beobachteten Subjekten aufgerufen, so bald diese sich verändern. Da beim Aufruf der Methode das überwachte Element mit übergeben wird, ist es für die Observer nicht zwingend notwendig, eine Referenz auf das zu überwachende Objekt vor zuhalten.

Zur Verwaltung der *Observer*-Objekte in den zu überwachenden Subjekten, kommt die C# spezifische Besonderheit des *Delegate* zum Einsatz. Diesem *Delegate* kann jeweils die entsprechende *ReceiveObserverUpdate()*-Methode des überwachenden Objektes hinzugefügt und wieder entfernt werden.

5.3. Design der Anwendung

Die Anwendung EvoCSS soll eine endliche Anzahl von Stylesheets vorhalten, eine Auswahl daraus dem Benutzer zur Bewertung präsentieren und basierend auf der Nutzverwertung eine, besser auf den Nutzer abgestimmte, Generation neuer Stylesheets erzeugen. Die Stylesheets setzen sich dabei aus CSS-Divisionen und diese wieder aus CSS-Attributen zusammen. Diese CSS-Attribute sollen unabhängig ihrer tatsächlichen Ausprägung polymorph behandelt werden. Hierzu kommen einheitliche Schnittstellen zum Einsatz.

Die Darstellung der einzelnen Elemente und Einstellung der Programmparameter erfolgt über eine grafische Oberfläche. Mit deren Hilfe hat der Nutzer die Möglichkeit einzelne Individuen zu bewerten und Programmparameter zu manipulieren.

Es folgt eine Übersicht aller Programmbibliotheken, mit einer kurzen Beschreibung der Funktionalitäten.

5.3.1. Bibliothek EvoCSS-Values – Die CSS-Werte

Alle CSS-Werte sind von der abstrakten Oberklasse *CssValue* abgeleitet. Diese stellt abstrakte Methoden und Properties zur Verfügung, die jede abgeleitete Klasse im Minimum implementieren muss. So wird sicher gestellt, dass sich die CSS-Werte im Programm gleich behandeln lassen. Jeder Werttyp muss z.B. in der Lage sein, seine CSS-Repräsen-

tation als String zurückzugeben, oder seinen Eigenschaftswert zu setzen bzw. zurückzugeben. Zur Erweiterung des Programms und der Verarbeitung komplexerer Stylesheets, mit mehr Werttypen, können an dieser Stelle neue Werttypen von der abstrakten Oberklasse *CssValue* abgeleitet werden. Die folgende Abbildung zeigt das Klassendiagramm der Bibliothek EvoCSS-Values. Aus Platzgründen sind einige Member ausgeblendet.

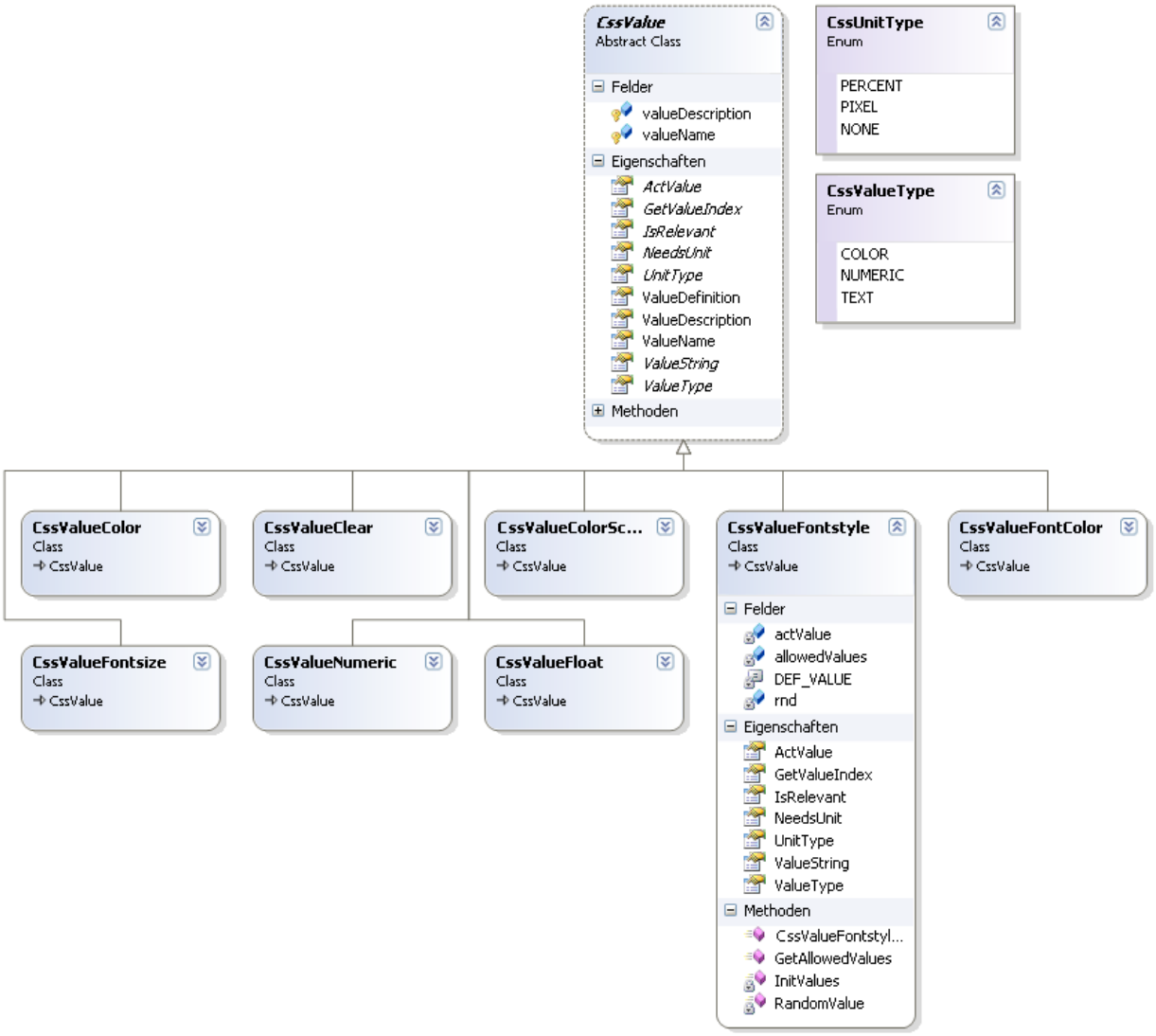

Abbildung 8: Übersicht der Bibliothek EvoCSS-Values

5.3.2. Bibliothek EvoCSS-Elements – Die Elemente der Applikation

In der Bibliothek EvoCSS-Elements sind die wichtigsten Klassen zur Strukturierung der Applikation zu finden. Die Klasse *Webpage* realisiert dabei die eigentliche HTML-Seite und stellt Methoden zu Erzeugung des wohl geformten HTML-Codes und der Vorschaudateien

bereit. Die Klasse *Stylesheet* implementiert die eigentlichen Individuen des Populationspools und stellt wichtige Methoden zum Vergleich, Clonen oder zur zufälligen Erzeugung eines Individuums bereit. Zum Vergleich wird ein *Comparator*-Schnittstelle zur Verfügung gestellt, so lassen sich beliebige Vergleichsoperationen für die einzelnen *Stylesheet*-Objekte definieren, eine Erweiterung um zusätzliche Vergleichsoperatoren ist problemlos möglich. Jedes Stylesheet besteht dabei aus CSS-Divisionen, diese werden durch die Klasse *CssDivision* repräsentiert.

Weiterhin finden sich in dem Paket die Klasse *Observable* und das Interface *ObserverIF*, diese realisieren die benötigen Schnittstellen für das Beobachtermuster. Alle Beobachter müssen das Interface *ObserverIF* implementieren, alle zu beobachtenden Subjekte müssen von der konkreten Oberklasse *Observable* abgeleitet werden. Eine genauere Betrachtung der Implementierung des Beobachtermusters, mit Hilfe eines *Delegate-Events,* folgt im nächsten Kapitel.

Außerdem ist das Interface *ApplicationIF* in der Bibliothek enthalten, dieses stellt Schnittstellen für Basisfunktionen der eigentlichen Applikation zur Verfügung, um die Funktionen direkt aus dem Evolutionsprozess anstoßen zu können. Die folgende Abbildung zeigt das Klassendiagramm der Bibliothek EvoCSS-Elements, wobei aus Platzgründen einige Member ausgeblendet wurden. Die kompletten Klassendiagramme aller Bibliotheken sind auf der, der Arbeit beiliegenden, CD enthalten.

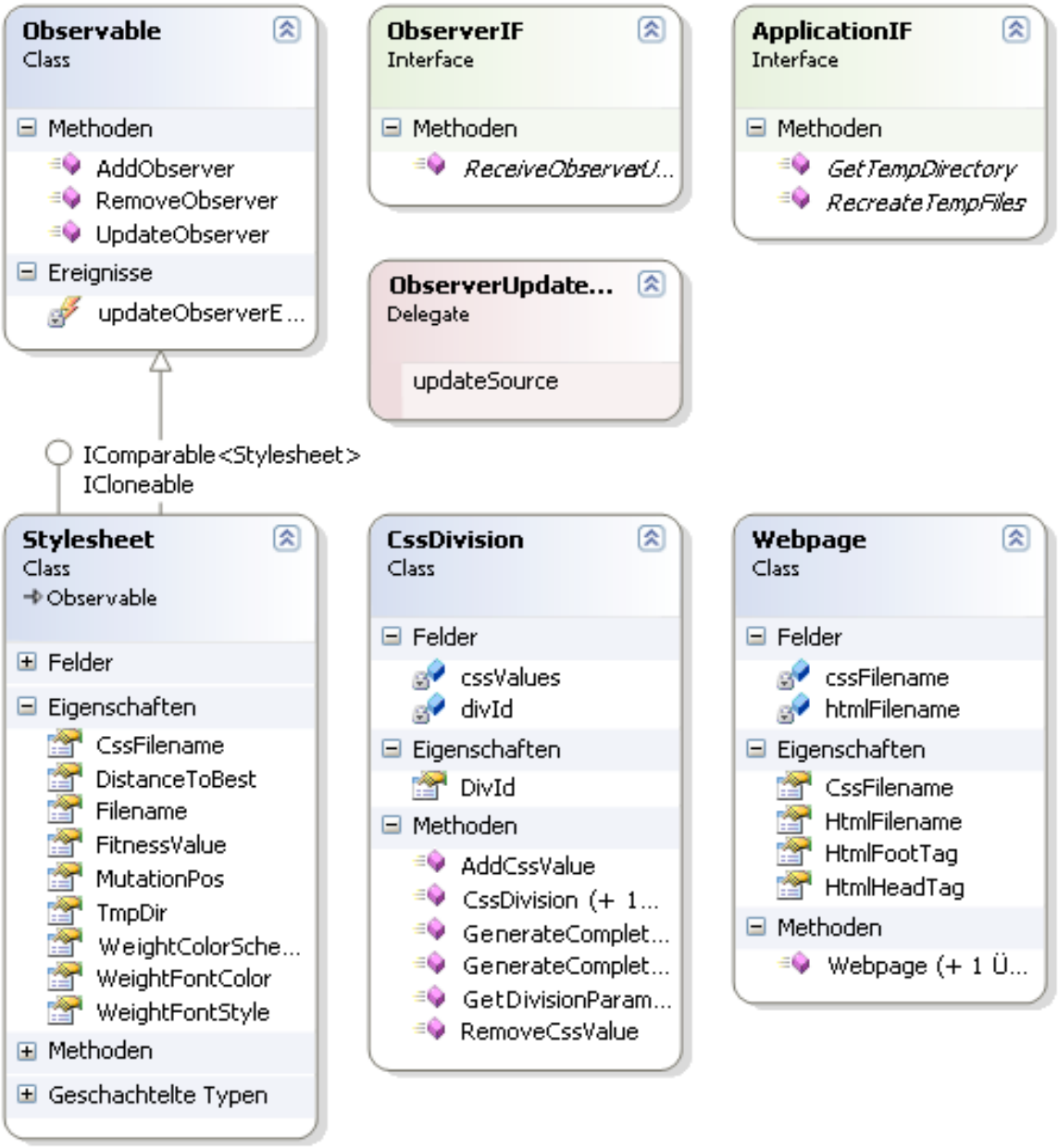

Abbildung 9: Übersicht der Bibliothek EvoCSS-Elements

5.3.3. Bibliothek EvoCSS-DataAccess – Lesen und Schreiben von Dateien

Die Bibliothek EvoCSS-DataAccess enthält nur eine einzige Klasse *FileIO*, diese realisiert das Lesen und Schreiben von Dateien. Die Klasse stellt dabei Methoden zur Verfügung, den kompletten CSS- und HTML-Code einer Darstellung zu Erzeugen und in entsprechende Dateien zu schreiben. Die folgende Abbildung zeigt das Klassendiagramm des Paketes.

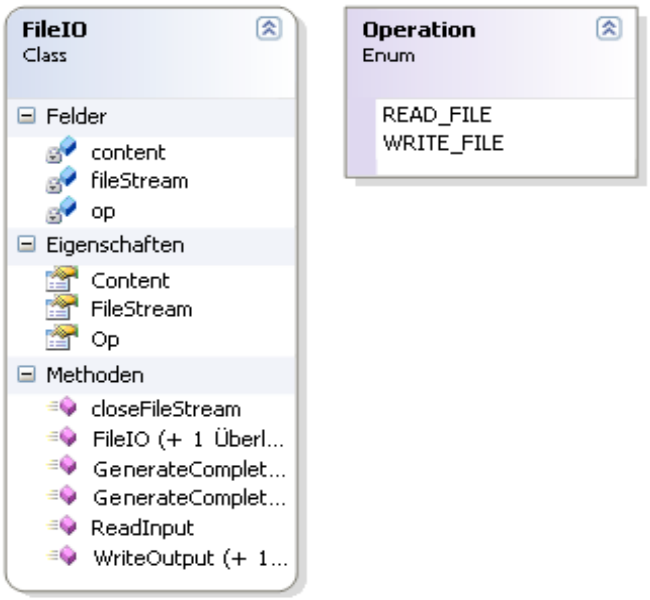

Abbildung 10: Übersicht der Bibliothek EvoCSS-DataAccess

5.3.4. Die Bibliothek EvoCSS-Exceptions – Ausnahmebehandlung

Die Applikation enthält in der vorliegenden Version ein sehr begrenztes Exception Handling. Das Paket enthält daher nur zwei benutzerdefinierte Exceptions, die *ValueNotAllowedException* wird durch das Programm geworfen, sobald versucht wird einen CSS-Wert mit Parametern außerhalb seines Wertebereichs zu definieren. Die *TempDirNotCreatedException* wird geworfen, wenn im Programmverlauf versucht wird temporäre Dateien zu speichern, aber klein gültiges temporäres Verzeichnis vorhanden ist. Die Abbildung zeigt das Klassendiagramm des Paketes.

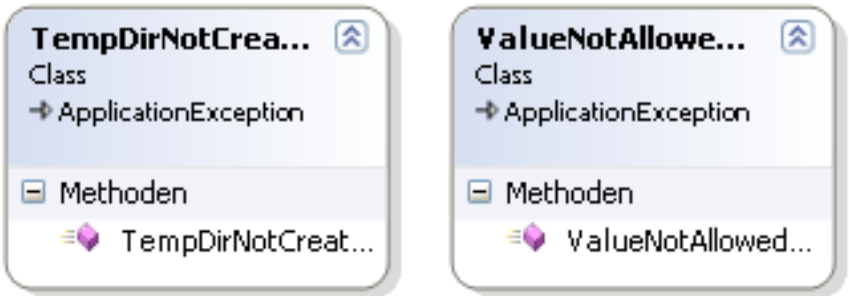

Abbildung 11: Übersicht der Bibliothek EvoCSS-Exceptions

5.3.5. Die Bibliothek EvoCSS-Evolution – Die eigentliche Evolution

Dieses Paket enthält nur eine Klasse: *EvolutionClass*, diese Klasse implementiert die eigentlichen Funktionen der Evolution. So werden Methoden zur Verfügung gestellt um eine Startpopulation zu erzeugen, Individuen zu selektieren oder die Evolution zu reinitialisieren. Eine detaillierte Beschreibung der Funktionsweise kann dem Kapitel Implementierung entnommen werden.

5.3.6. Die Bibliothek EvoCSS-Controls – Die modularen Steuerelemente

Die zentralen Steuerelemente der Anwendung wurden einzeln, als so genannte *User-Controls*, implementiert. Diese einzelnen *User-Controls* können modular auf die Hauptanwendung aufgesetzt werden. Alle in diesem Paket enthaltenen *User-Controls* übernehmen auch Aufgaben der Darstellung, daher implementieren sie alle das Interface *ObserverIF*, um auf Änderungen der, durch sie überwachten, Subjekte mit entsprechenden Aktualisierungen zu reagieren.

Die Klasse *ControlPanel* implementiert den linken Bereich der Applikation, mit allen benötigten Steuerelementen und Eingabemöglichkeiten. Die Klasse *BrowserElement* realisiert ein einzelnes Vorschauelement, es stellt die Anzeige eines Individuums als HTML-Ansicht zur Verfügung. Weiterhin wird die Fitness des Individuums ausgegeben und es werden zwei Schaltflächen zur Bewertung der Fitness, des angezeigten Individuums zur Verfügung gestellt. Die Klasse *BrowserPanel* fasst neun *BrowserElement User-Controls* zu einem großen Panel zusammen. Durch den modularen Aufbau kann die Anzahl der anzeigten Individuen sehr leicht angepasst werden, da nur zusätzliche *User-Controls* auf das Panel aufgesetzt werden müssen. Zur Darstellung der einzelnen Individuen kommt die Klasse *WebBrowser* zum Einsatz, die durch die .net-Umgebung zur Verfügung gestellt wird.

Die folgenden beiden Abbildungen zeigen die Design-Ansicht der beiden *User-Controls BrowserElement* und *ControlPanel*, um den modularen Aufbau der Steuerelemente zu verdeutlichen. Direkt anschließend findet sich eine Darstellung des Klassendiagramms der Bibliothek EvoCSS-Controls, aus Platzgründen wurden einige Member ausgeblendet.

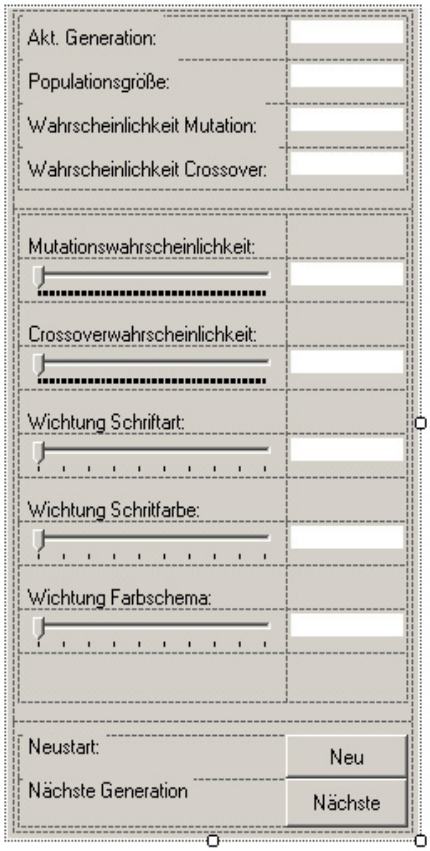

Abbildung 12: UserControl ControlPanel, enthält sowohl Eingabe- als auch Ausgabeelemente

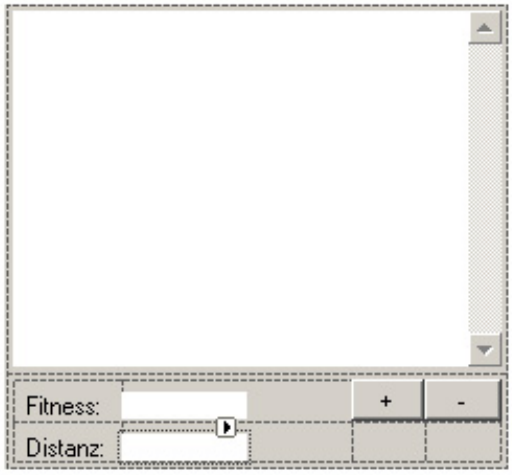

Abbildung 13: UserControl - BrowserElement, 9 dieser Elemente werden zu einem BrowserPanel zusammengesetzt

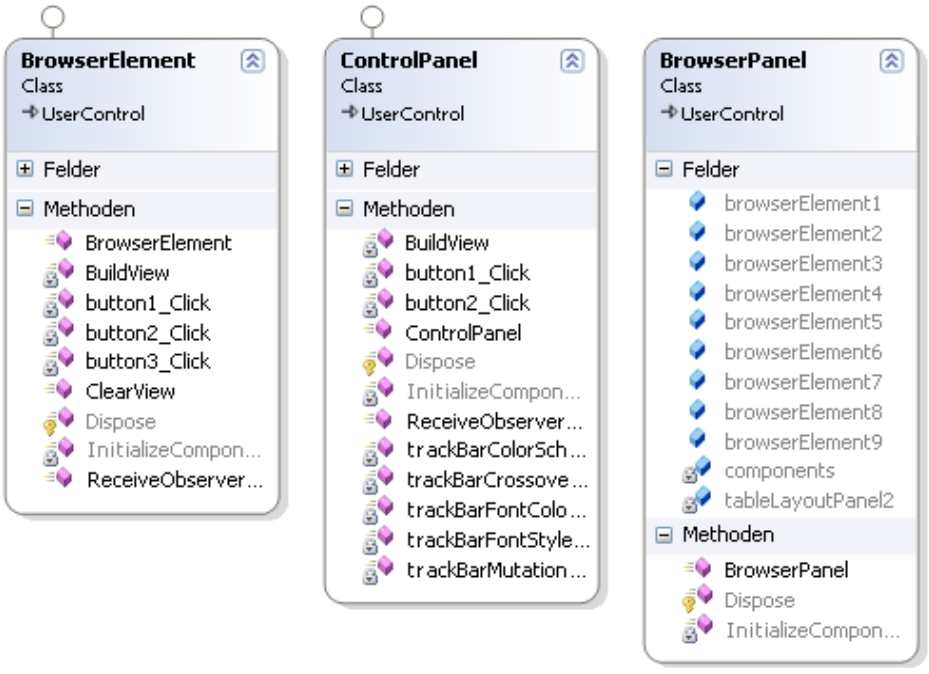

Abbildung 14: Übersicht der Bibliothek EvoCSS-Controls

5.3.7. EvoCSS – Die Hauptanwendung

Die Klasse *FrmMain* realisiert das eigentliche Hauptanwendungsfenster. Hier werden zentral alle benötigten Objekte vorgehalten und die entsprechenden Methoden aufgerufen. Die Klasse bietet weiterhin wichtige Methoden, die z.B. beim Schließen der Applikation das temporäre Verzeichnis ordnungsgemäß löschen, oder eine Neuerstellung aller Vorschaudateien anstoßen können. Weiterhin werden die, durch das Programm verwendeten, Startparameter hier zentral als Konstanten abgelegt. Die *FrmMain* Klasse initialisiert die einzelnen Viewelemente und ist verantwortlich für die An- und Abmeldung der entsprechenden *Observer*-Objekte.

Zusätzlich wird durch die *FrmMain*-Klasse das Interface *ApplicationIF* implementiert, dieses stellt zwei Methoden zum Auslesen des temporären Verzeichnisses und der Neuerstellung der temporären Dateien zur Verfügung. Es dient dabei als Schnittstelle für die Klasse *EvolutionClass*.

5.4. Die Implementierung

In diesem Kapitel wird die eigentliche Implementierung der Anwendung beschrieben, zentrale Klassen und Methoden werden detailliert vorgestellt und deren Zusammenarbeit erläutert. Der Fokus liegt dabei auf den Bereichen, die direkt mit dem evolutionären Ansatz der Anwendung zusammenhängen.

5.4.1. Die Repräsentation der Individuen

Lange Zeit wurde für die Anordnung der unterschiedlichen Bereiche einer Webseite eine Tabellenstruktur verwendet, so war es möglich die Elemente relativ einfach auf dem Bildschirm anzuordnen. Programmiersprachen wie Java oder C# verwenden ähnliche Konzepte um die Elemente von grafische Oberflächen anzuordnen. Tabellen haben allerdings den großen Nachteil, dass sie die Barrierefreiheit von Webseiten stark einschränken. Genauer betrachtet sind Tabellen auch kein Element zur inhaltlichen, sondern lediglich zur optischen Strukturierung eines Dokuments. Per Konvention sollte die optische Gestaltung allerdings nicht im HTML-Code vorgenommen werden. Moderne Webseiten verwenden daher ein Layout auf Basis von so genannten DIV-Containern, die auch als CSS-Divisionen

bezeichnet werden. Der englische Begriff, für den es leider keine eindeutige deutsche Entsprechung gibt, lautet "division".

Die Divisionen bieten die Möglichkeit bestimmte Bereiche von Webseiten zusammenzufassen und Ihre Darstellung und Anordnung zu steuern. Für jede Division können eigene Stilinformationen definiert werden, oder es werden die Definitionen des umschließenden Elementes geerbt. Die Anwendung verfolgt ebenfalls diesen Ansatz von tabellenfreien Web-Layouts und fasst die einzelnen Bereiche, in der vorliegenden Version zu vier Divisionen, zusammen. (Kopf, Navigation, Content, Fuß) Die einzelnen Divisionen enthalten CSS-Attribute und werden ihrerseits wieder zu Stylesheets zusammengefasst.

5.4.1.1 Die Klasse CssValue und abgeleitete Elemente

Die abstrakte Oberklasse *CssValue* realisiert die einzelnen CSS-Parameter, von ihr werden weitere konkrete CSS-Werte abgeleitet. Grundsätzlich besteht ein CSS-Wert aus einem Namen, einem Wert und, so weit benötigt, einer Einheit, dabei können drei Haupttypen unterschieden werden: Farbwerte, numerische Werte und String-Werte. Die wichtigste konkrete Methode innerhalb der Klasse,ist die Methode *ValueDefinition()*, sie liefert einen gültigen CSS-String zurück, der direkt für das Schreiben der CSS-Dateien verwendet wird. Einige abgeleitete Klassen, z.B. *CssValueFontColor* überschreiben diese Methode um ihren spezifischen Eigenschaften gerecht zu werden, im Falle von Farben müssen die *Color*-Objekte erst in die HTML-typische hexadezimale Schreibweise umgeformt werden.

Jede konkrete Ableitung von *CssValue* enthält ein *Array* oder eine *Arraylist allowedValues*, in diesen Listen werden alle zulässigen Werte vorgehalten. Die einzelnen Klassen sind so sehr schnell um neue Parameterwerte erweiterbar. Um diese Werte über ihren konkreten Typ hinaus und über die Grenzen der Division vergleichen zu können, sind diese Arrays geordnet. Schriftgrößen wurden von der Kleinsten zur Größten, Farben an Hand ihrer Position im Farbkreis und Schriftarten nach ihrem Typ, zu geordneten Listen zusammengefasst. Jede von *CssValue* abgeleitete Klasse muss die abstrakte Property *GetValueIndex* realisieren, um den Index des aktuellen Wertes zurückgeben zu können.

Nicht jeder CSS-Wert soll Eingang in den evolutionären Prozess finden, daher enthält *CssValue* die abstrakte Property *IsRelevant,* die von allen abgeleiteten Klassen überschrieben werden muss. Die Klassen können so über ein *Boolean* zurückgeben, ob sie mit in die Evolution einbezogen werden oder nicht. Werte, wie die Ausrichtung, das Umbruchsverhalten oder die Größe von Containern, werden im vorliegenden Programm nicht

berücksichtigt.

Eine weitere wichtige Property ist *ActValue*. Diese wird ebenfalls als abstrakte Property in *CssValue* definiert, mit Hilfe dieser Eigenschaft werden die konkreten Werte gesetzt und ausgelesen. Um alle *CssValue* Objekte gleich behandeln zu können, ist der Übergabetyp der Property *Object*. Alle abgeleiteten Klassen müssen sicher stellen, dass zum Einen nur Werte des konkreten Typs, zum anderen nur Werte innerhalb des zulässigen Definitionsbereiches, also Werte die sich in der Liste *allowedValues* befinden, gesetzt werden dürfen. Sollte sich der zu setzende Wert nicht in der Liste befinden, wird eine *ValueNotAllowedException* geworfen.

5.4.1.2 Die Klasse CssDivision

Alle zu einem Container gehörenden CSS-Werte werden in einem *CssDivision*-Objekt zusammengefasst. Die einzelnen enthaltenen *CssValue*-Objekte werden als typisierte *Arrayliste* vorgehalten. Die *CssDivision*-Klasse stellt unterschiedliche Methoden zur Verfügung um Attribute hinzuzufügen, den kompletten CSS- und HTML-Code eines Containers zu erzeugen und den Parametervektor eines Containers zurückzugeben. Bei der Methode *GetParamVector* kommt ein *Dictionary* als Rückgabetyp zum Einsatz, dieses erlaubt es die einzelnen CSS-Werte über eine "Key-Value"-Kombination abzulegen. Als Schlüssel dient der Name des Attributes, der Wert entspricht der Indexposition des aktuell gesetzten Wertes in der Liste *allowedValues*.

5.4.1.3 Die Klasse Stylesheet

Die Klasse *Stylesheet* repräsentiert die konkrete Ausprägung eines Individuums, sie ist direkt von *Observable* abgeleitet und bietet damit für die Browserelemente eine Schnittstelle sich als Beobachter anzumelden.

Das typisierte Interface *IComparable<Stylesheet>* wird durch die Klasse implementiert, um Vergleichsoperationen für *Stylesheet*-Objekte zur Verfügung zu stellen. Zum direkten Vergleich von zwei Stylesheets wird die Methode *CompareTo()* einmal mit einem Stylesheet als Übergabeparameter definiert, dies entspricht der durch das Interface geforderten Schnittstelle. Für den Vergleich wird hier der Fitnesswert herangezogen. Die erweiterte *CompareTo()*-Methode nimmt als Übergabeparameter ein Stylesheet und einen Komparatortyp entgegen, so ist sicher gestellt das die Funktionalitäten zum Vergleich leicht angepasst und erweitert werden können.

Um diese erweiterte Vergleichbarkeit zu realisieren kommt die interne Klasse *Stylesheet-Comparer* zum Einsatz. Sie hält in einem *Enum* die möglichen Komparatortypen vor, in der vorliegenden Programmversion kann ein Vergleich auf Grundlage der Fitness, oder auf Grundlage des Abstandes zweier Parametervektoren durchgeführt werden. Konkret eingesetzt wird die Funktionalität um die Listen von Stylesheets zu sortieren, dazu wird erst ein Objekt der Klasse *StylesheetComparer* erzeugt, diesem muss der gewünschte Vergleichstyp übergeben werden, anschließend kann das *Comparer*-Objekt direkt an die *Sort()* Methode der *Arrayliste* übergeben werden.

Zusätzlich wird durch die Klasse *Stylesheet* das Interface *IClonable* implementiert, dieses verlangt die Realisierung der *Clone()*-Methode mit dem Rückgabetyp *Object*. Die Methode erzeugt ein neues *Stylesheet*-Objekt, anschließend werden alle Aktualparameter des Objektesm an dem die Methode aufgerufen wurde, in das neue Objekt kopiert und die Referenz auf das geklonte Objekt zurückgegeben.

Eine der wichtigsten Methoden die die Klasse *Stylesheet* bereitstellt, ist die Methode *GenerateParamVektor()*. Diese gibt eine typisierte *Arrayliste* mit *Integer*-Werten zurück um das Stylesheet zu beschreiben, sie ruft dazu die Methode *GetDivisionParamVector()* der einzelnen Container auf, diese wiederum rufen die Property *GetValueIndex* für die einzelnen enthaltenen CSS-Attribute auf. Der Parametervektor hat in der vorliegenden Programmversion die Länge 10, die ersten beiden Werte beschreiben die Farbkombination und die Schriftfamilie. Diese Werte werden auf Grund eines einheitlichen Erscheinungsbildes für alle CSS-Container nur einmal gesetzt. Die danach folgenden Werte entsprechen jeweils abwechselnd der Schriftgröße und Schriftfarbe der vier Container "head", "navi", "content" und "foot".

Auf Grundlage dieses Parametervektors kann der Abstand zwischen zwei Ausprägungen eines *Stylesheet*-Objektes bestimmt werden. Zur konkreten Bestimmung des Abstandes kommt die Methode *CalculateParamDistance()* zum Einsatz, als Übergabewert erwartet sie ein weiteres *Stylesheet*-Objekt. Der Rückgabewert ist ein *Integer*, welcher den Abstand zwischen dem *Stylesheet*-Objekt an dem die Methode aufgerufen und dem übergebenen *Stylesheet*-Objekt darstellt. Hierzu wird von jedem Wert des Parametervektors die Differenz gebildet und das Quadrat gebildet, die Ergebnisse werden aufsummiert und bilden in Summe den Abstand. Um bestimmte Attribute stärker oder weniger stark zu berücksichtigen kommen Wichtungsfaktoren zum Einsatz. Die Wichtungsfaktoren *WeightColorScheme*, *WeightFontStyle* und *WeightFontColor* können ganzzahlige Werte zwischen 1 und 10

annehmen. Sie werden direkt mit der quadrierten Summe des jeweiligen Vektorindex multipliziert und bieten so die Möglichkeit die einzelnen Attribute zu wichten.

Die Methode *CreateFromParamVector()* bietet die Funktionalität ein *Stylesheet*-Objekt direkt aus einem Parametervektor zu erzeugen, wichtigster Übergabeparameter ist der Parametervektor. Aus diesem erzeugt die Methode ein neues *Stylesheet*-Objekt und gibt die Referenz auf dieses entsprechend zurück.

Die Crossoverfunktionalität wird durch die Methode *CalculateMediumParamVector()* realisiert, auch sie erwartet ein anderes Stylesheet als Übergabeparameter. Die Methode berechnet, für jeden Parameterwert des übergebenen *Stylesheet*-Objektes und des *Stylesheet*-Objekt an dem sie aufgerufen wurde, jeweils den Mittelwert. Das Ergebnis ist dabei wieder ein Parametervektor, der ein *Stylesheet*-Objekt beschreibt, welches im Sinne das Abstandes, zwischen den beiden verglichenen *Stylesheet*-Objekten liegt. Dieser Vektor kann direkt an *CreateFromParamVector()* übergeben werden, um ein Individuum zu instantiieren.

Die Funktionalitäten für die Mutation werden durch die Methode *Mutate()* implementiert, hierzu wird zuerst eine gleichverteilte Zufallszahl zwischen 1 und 100 erzeugt, sollte die erzeugte Zufallszahl kleiner oder gleich der Mutationswahrscheinlichkeit in Prozent sein wird der Parametervektor mutiert. Dazu wird für jeden Index des Vektors eine Zufallszahl zwischen 0 und 1 ermittelt, bei 1 wird der Wert inkrementiert, bei 0 wird der Wert dekrementiert. Aus dem erzeugten Parametervektor wird durch Aufruf der Methode *CreateFromParamVector()* ein neues *Stylesheet*-Objekt erzeugt und zurückgegeben. Sollte die erzeugte Zufallszahl für die Mutation größer der Mutationswahrscheinlichkeit in Prozent sein, wird aus dem ursprünglichen Vektor ein neues *Stylesheet*-Objekt erzeugt und zurückgegeben.

5.4.2. Der evolutionäre Kreislauf

Die Klasse *EvolutionClass* realisiert die Aufgaben der eigentlichen Evolution, mit der Instantiierung eines neuen Objektes, wird durch die Methode *InitStartingPop(),* eine neue Population von Individuen erzeugt. Die Startparameter sind im Hauptprogramm hinterlegt und werden dem Konstruktor übergeben. Hierzu zählen beispielsweise die Größe des Populationspools, die Anzahl der Vorschauindividuen und die Mutations- sowie Crossoverwahrscheinlichkeit. Nachdem die Startpopulation zufällig erzeugt wurde bietet die Klasse mit der Methode *GetPreviewIndividuals()* eine Schnittstelle um die definierte Anzahl der

Vorschauindividuen als Liste von *Stylesheet*-Objekten zurückzugeben. Die Methode geht von einer geordneten Population aus und gibt immer die ersten Individuen der Population zurück. Die Methode *ResetEvolution()* dient dazu den gesamten Evolutionsprozess zurückzusetzen, dazu wird eine neue zufällige Startpopulation erzeugt und alle Parameter der Evolution auf ihre Standardwerte zurückgesetzt.

EvaluateGenStep() ist eine weitere wichtige Methode innerhalb der Evolutionsklasse, diese Methode wird aufgerufen so bald der Nutzer auf die Schaltfläche "Nächste" drückt und die nächste Generation erzeugt werden soll. Im ersten Schritt wird hier ein *Comparer*-Objekt erzeugt und die komplette Population nach Fitness geordnet. Das aktuell bestbewertete Individuum mit der höchsten Fitness wird als Eliteindividuum gespeichert. Anschließend wird der Abstand aller in der Population enthaltenen Individuen zum Eliteindividuum bestimmt und entsprechend in den Objekten gespeichert. Auf Grundlage dieses Abstands greift nun ein Regelsystem, dass allen Individuen denen keine Fitness zugewiesen wurde einen Standardfitnesswert zuweist. Da nicht alle Individuen des Populationspools zur Anzeige kommen können, um durch den Nutzer bewertet zu werden, ist dieser Schritt der Versuch, abhängig vom durch den Nutzer gewählten Eliteindividuum, eine Fitnessabschätzung vorzunehmen.

Das Regelsystem zur Fitnesszuweisung teilt den Abstand der Individuen in Intervalle ein, wenn der Abstand größer 1000 ist erhält das Objekt standardmäßig 1% der Fitness des Eliteindividuum, bis 500 erhält es 10%, bis 250 werden 25% zugewiesen, sollte der berechnete Abstand kleiner 250 sein, wird dem Individuum eine Standardfitness von 50% der Fitness des Eliteindividuums zugewiesen. Nachdem allen Individuen des Pools die nicht durch den Nutzer bewertet wurden, eine Standardfitness zugewiesen wurde, beginnt die eigentliche Selektion der Individuen. Für die Selektion wurde die Rouletteselektion implementiert, hierzu werden alle Individuen so oft, wie ihr Fitnesswert hoch ist, in eine neue *Arrayliste roulette* eingefügt. Ein Individuum mit einem Fitnesswert von 20, würde also 20 mal in die Liste eingefügt werden. Anschließend wird eine gleichverteilte Zufallszahl im Bereich von 0 bis Summe aller Fitnesswerte der Liste ermittelt. Dieser ermittelte Wert dient direkt als Index um ein Individuum aus der Liste *roulette* zurückzugeben. Die Rouletteselektion selektiert mit einem Aufruf immer genau ein Individuum.

Bevor jedoch die eigentlich Selektion stattfindet, wird erst überprüft ob es zu einem Crossover kommt, dazu wird wieder eine gleichverteilte Zufallszahl zwischen 0 und 100 ermittelt, ist die Zufallszahl kleiner oder gleich der Crossoverwahrscheinlichkeit werden durch

zweimaliges Aufrufen der Selektionsmethode zwei Individuen als Eltern selektiert. Mit Hilfe der *CalculateMediumParamVector()*-Methode aus *Stylesheet,* wird aus den beiden selektierten Elternindividuen eine Kind erzeugt und nachdem es eventuell mutiert wurde, in die neue Generation eingefügt. Sollte die Zufallszahl größer der Crossoverwahrscheinlichkeit sein, findet kein Crossover statt, stattdessen wird nur ein Individuum selektiert, anschliessend eventuell mutiert und in die neue Generation eingefügt. Der Vorgang dauert so lange an bis die neue Generation die selbe Größe erreicht hat, wie die Elterngeneration. Das Eliteindividuum wird grundsätzlich in die neue Generation übernommen. Nun wird durch die *EvaluateGenStep()*-Methode das Neuerstellen der Vorschaudateien angestoßen, die Darstellung auf dem Bildschirm wird aktualisiert und der Prozess beginnt von Neuem.

Die Klasse *EvolutionClass* ist von *Observable* abgeleitet und bietet damit die Schnittstelle für ein *Observer*-Objekt sich entsprechend anzumelden, um auf Änderungen zu reagieren.

5.4.3. Das Beobachtermuster

Die Basisfunktionalitäten für das Beobachtermuster werden durch die Klasse *Observable* und das Interface *ObserverIF* bereit gestellt. Das *ObserverIF* verlangt von allen Beobachtern im Minimum die Implementierung der Methode *ReceiveObserverUpdate(Observables)*, als Übergabeparameter enthält die Methode das zu überwachende Subjekt. Die Methode wird durch die überwachten Subjekte bei Änderungen angestoßen.

Die *Observable*-Klasse bietet zum Speichern der einzelnen Beobachter ein *Delegate-Event*, diesem *Delegate* können Methoden hinzugefügt und wieder entfernt werden. Die beiden Methoden *AddObserver(ObserverIF o)* und *RemoveObserver(ObserverIF o)* bieten genau diese Funktionalität. Sie fügen die *ReceiveObserverUpdate()* Methode, des übergebenen Beobachters direkt dem *Delegate* hinzu, bzw. entfernen diese. Sollte ein beobachtetes Subjekt sich ändern ruft es die *UpdateObserver()*-Methode auf, diese löst den *Delegate-Event* aus und informiert so alle angemeldeten Beobachter über die Änderungen.

Im vorliegenden Programm implementieren die Klassen *BrowserElement* und *ControlPanel* das *ObserverIF*-Interface und stellen damit Beobachter Funktionalität zur Verfügung. Die Klassen *Stylesheet* und *EvolutionClass* werden von *Observable* abgeleitet und bieten die Funktionalität für überwachte Subjekte. Zum Programmstart und mit jeder neuen Generation, werden die neun Browserelemente an den Stylesheets angemeldet, die durch *GetPreviewIndividuals()* von der Evolutionsklasse bereitgestellt werden.

Die Funktionalität zum An- und Abmelden der Browserelemente findet sich im Hauptprogramm in den beiden Methoden *AddObserver()* und *RemoveObserver()*. Zusätzlich wird durch die beiden Methoden auch das Controlpanel an der Evolutionsklasse angemeldet. So wird sicher gestellt, dass nach Programmstart und nach Erzeugung einer neuen Generation, alle Beobachter an den korrekten zu beobachtenden Subjekten angemeldet sind.

6. Anwendungsbeschreibung

Nach dem Start befindet sich die Applikation im Initialisierungszustand, alle Programmparameter sind auf die, in *FrmMain* definierten, Standardwerte gesetzt. Das Programm erzeugt zum Start eine neue Population und zeigt zufällig neun Individuen der Population an. Im linken oberen Bereich werden Informationen zur aktuellen Generation und den Parametern angezeigt. Unter anderem wird hier der aktuelle Generationsschritt, die Populationsgröße, die Mutations- und Crossoverwahrscheinlichkeit ausgegeben. Direkt unter der Ausgabe befinden sich zwei Schieberegler, um die Mutations- und Selektionswahrscheinlichkeit einzustellen. Die Änderungen werden mit dem nächsten Generationsschritt berücksichtigt.

Zwei weitere Schieberegler dienen dazu die Wichtungsfaktoren für Schriftart, -farbe und das Farbschema einzustellen. Die eingestellten Werte werden mit dem nächsten Generationsschritt übernommen und direkt, wie im Abschnitt 5.4.1.3 "Die Klasse Stylesheet" beschrieben, in den Parametervektor eingerechnet. Standardmäßig sind diese Werte auf 1 gesetzt und haben damit keinen Einfluss bei der Berechnung des Parametervektors. Unter diesen Schiebereglern befinden sich zwei Schaltflächen "Neu" und "Nächste". Die Schaltfläche "Neu" setzt die Applikation in den Startzustand zurück und erzeugt eine neue zufällige Generation.

Im rechten Bereich der Applikation wird eine Auswahl von neun Individuen aus dem Populationspool dargestellt. Zu jedem Individuum wird die aktuelle Fitness ausgegeben und ein weiteres Feld Distanz. Im Feld Distanz wird das Ergebnis der Abstandsbestimmung des Individuums zum derzeitigen Eliteindividuum ausgegeben. Rechts neben den beiden Ausgaben befinden sich zwei Schalter um die Fitness der jeweiligen Individuums zu inkrementieren oder zu dekrementieren.

7. Ungewöhnliche GUI-Konzepte

Im folgenden Kapitel werden sehr kurz einige ungewöhnliche Ansätze zur GUI-Gestaltung im Zusammenhang mit interaktiver Fitness dargestellt. Grundsätzliches Ziel aller gezeigten Konzepte ist es den Bewertungsprozess so spielerisch und wenig ermüdend, wie möglich zu gestalten, aber gleichzeitig eine möglichst hohe Auflösung der Bewertung zu erhalten.

7.1. Konzept Attributpyramide

Dieses Konzept ist am besten für die gleichzeitige Bewertung von bis zu drei Parametern geeignet, theoretisch wäre es auch möglich die Pyramide, im unteren Bereich des Bildschirmes, um weitere Attribute zu erweitern, doch ab mehr als drei Attributen können keine Schnittstellen zwischen den einzelnen Attributen mehr dargestellt und ausgewertet werden.

Die einzelnen Individuen fahren wie auf einem Laufband von rechts nach links über den Bildschirm. Die einfachste Bewertung ist die Bewertung "Neutral", dazu lässt der Nutzer das entsprechende Individuum nach links aus dem Bildschirm fahren. Alle Individuen die den Bildschirm verlassen, ohne das der Nutzer mit ihnen interagiert, werden als "Neutral" gewertet. Auf der Attributpyramide im unteren Bereich des Bildschirmes werden drei Zonen für mögliche Attribute markiert. Durch Drag&Drop hat der Nutzer die Möglichkeit ein Individuum vom Laufband zu nehmen und in einer Zone seiner Wahl wieder abzulegen.

Durch die Schnittstellen der drei Attribute auf der Pyramide, können alle drei Attribute mit einem einzigen Drag&Drop Vorgang bewertet werden. Wird das Individuum direkt in die Zone eines Attributs abgelegt, so wird nur dieses Attribut als "Gut" bewertet, wird das Individuum auf die Schnittstelle von zwei Attributen abgelegt (rote Kreis in der Darstellung) so werden diese beiden Attribute als "Gut" bewertet. Letztlich gibt es noch die Möglichkeit,

das Individuum direkt auf der Spitze der Pyramide abzulegen, so dass es in alle drei Bewertungsbereiche hineinragt. Diese Bewertung entspricht einem "Gut" für alle Attribute. Die folgende Abbildung veranschaulicht die Beschreibung durch eine kleine Skizze.

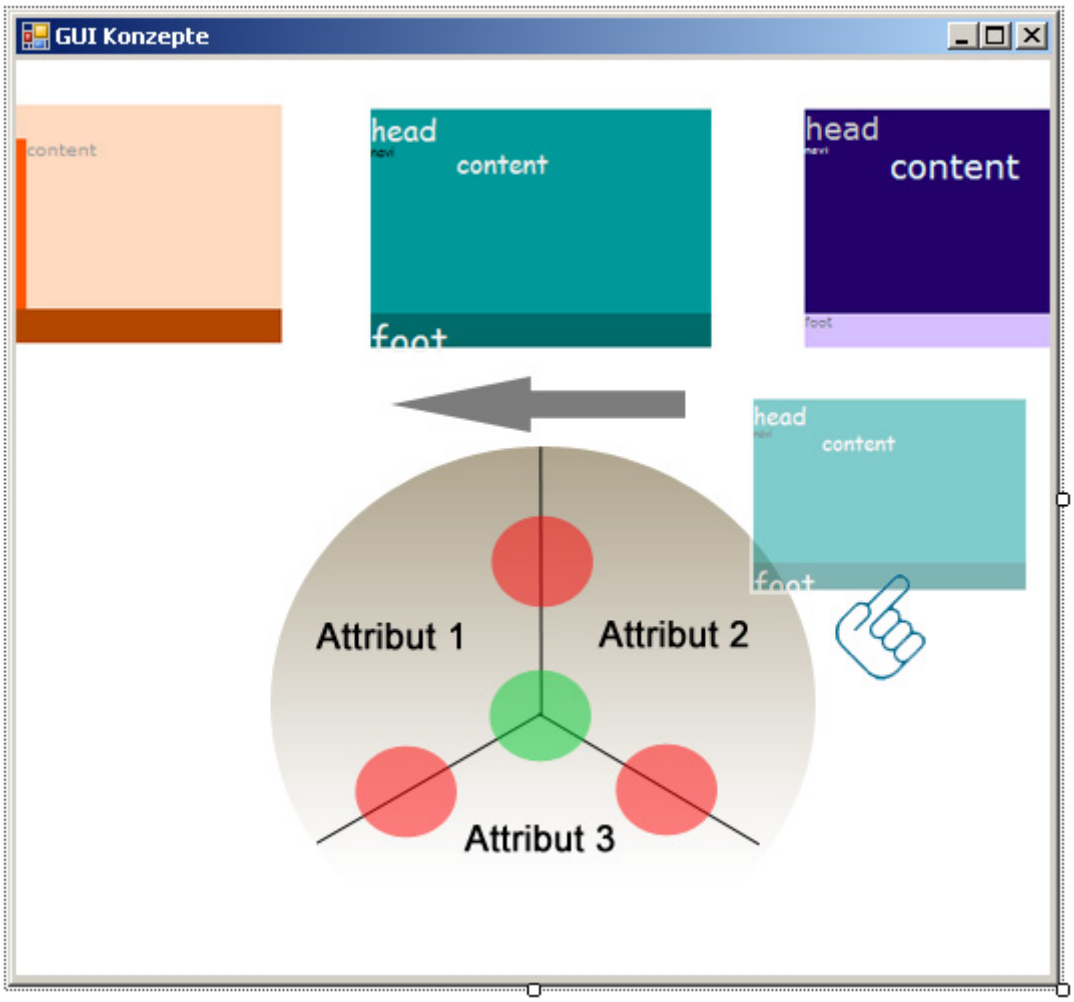

Abbildung 15: Bewertung der Indivduen mit Hilfe einer Attributpyramide

7.2. Konzept für mehrkriterielle Bewertung

Das Konzept der mehrkriteriellen Bewertung ist eine Erweiterung der Attributpyramide um mehr als drei Attribute gleichzeitig bewerten zu lassen. Gegenüber der Bewertung mittels Attributpyramide, hat das Konzept einen großen Nachteil, es lassen sich immer nur einzelne Parameter oder die gesamte Lösung bewerten. Eine Bewertung mehrerer Parameter gleichzeitig durch das Ablegen auf den Schnittstellen der Parameterbereiche ist nicht möglich.

Die einzelnen Individuen fahren wir im ersten Beispiel auf einem Laufband von rechts nach links über den Bildschirm. Einfachste Bewertung ist auch hier die Bewertung "Neutral", diese wird erteilt wenn das Individuum den Bildschirm verlässt ohne das der Nutzer mit ihm interagiert hat. Es gibt so viele Bewertungszonen wie zu bewertende Attribute, in der gewählten Abbildung sind vier Bewertungszonen enthalten. Der Nutzer hat wieder die Möglichkeit das Individuum per Drag&Drop vom Laufband aufzunehmen und in einem bestimmten Bereich abzulegen. Der grüne Kreis stellt die Zone dar in der alle Eigenschaften als positiv bewertet werden, der rote Kreis stellt eine Zone dar, in der alles als negativ bewertet wird. Die blauen Zonen beschreiben einzelne Attribute. Der Nutzer hat die Möglichkeit, genau ein Attribut der Lösung oder die gesamte Lösung als Einheit, zu bewerten. Die folgende Abbildung veranschaulicht das Konzept.

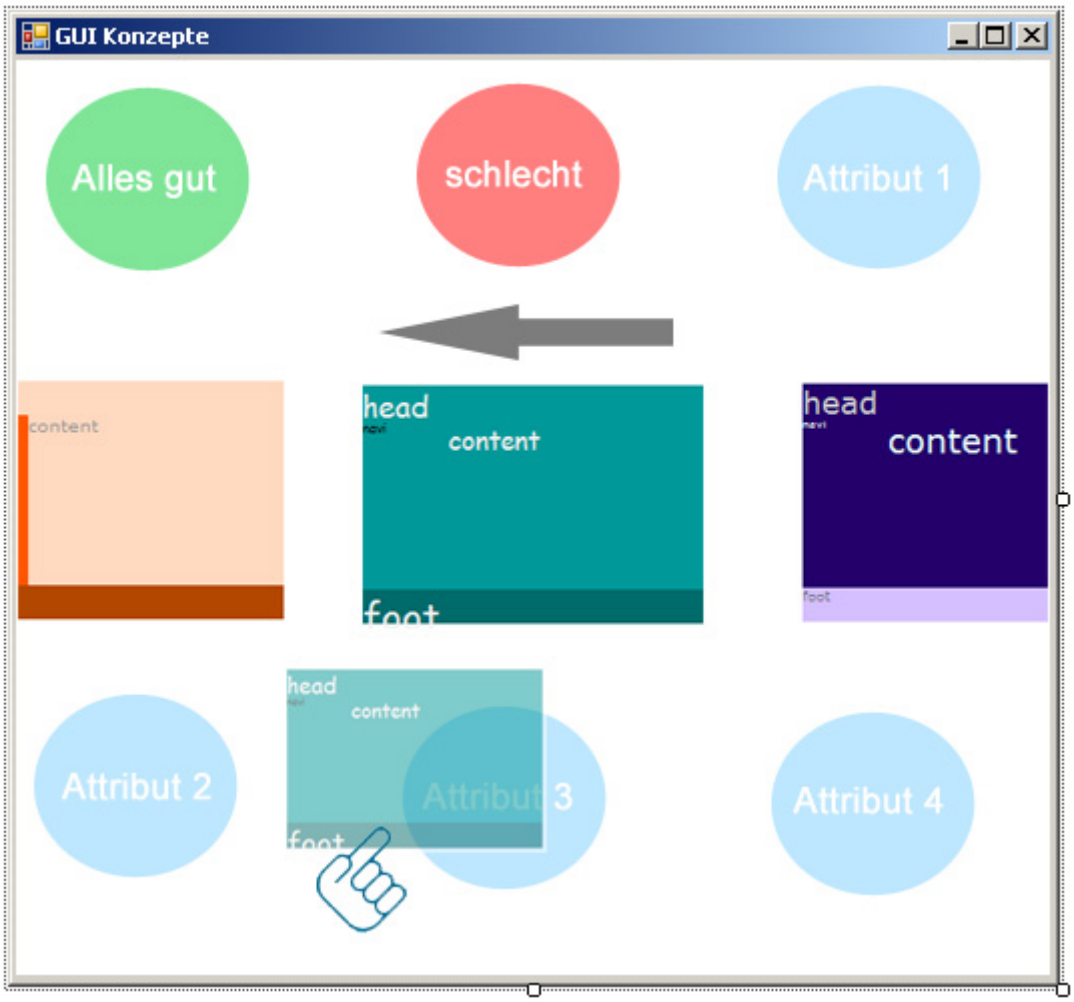

Abbildung 16: Mehrkriterielle Bewertung mit Hilfe von Attributzonen

7.3. Drei Klassen Modell

Dieses Konzept orientiert sich von der grundsätzlichen Funktionalität an den beiden ersten Konzepten. Die Bewertung ist allerdings weniger weit aufgelöst, dieser Ansatz erlaubt nur drei mögliche Bewertungen. Das Individuums wird dabei in seiner Gesamtheit bewertet, eine Wertung einzelner Parameter findet hier nicht statt. Die einzelnen Kandidaten fahren wieder auf einem Laufband von rechts nach links über den Bildschirm, Individuen die ohne Interaktion aus dem Bildschirm fahren werden als "Neutral" bewertet. Der Nutzer hat nun die Möglichkeit das Individuum als Ganzes, in eine der drei Klassen "Gut" (repräsentiert durch die Zone mit der grünen Ellipse), "Schlecht" (repräsentiert durch die rote Zone) sowie "Neutral" einzuordnen. Die dargestellte Abbildung verdeutlicht diesen Ansatz.

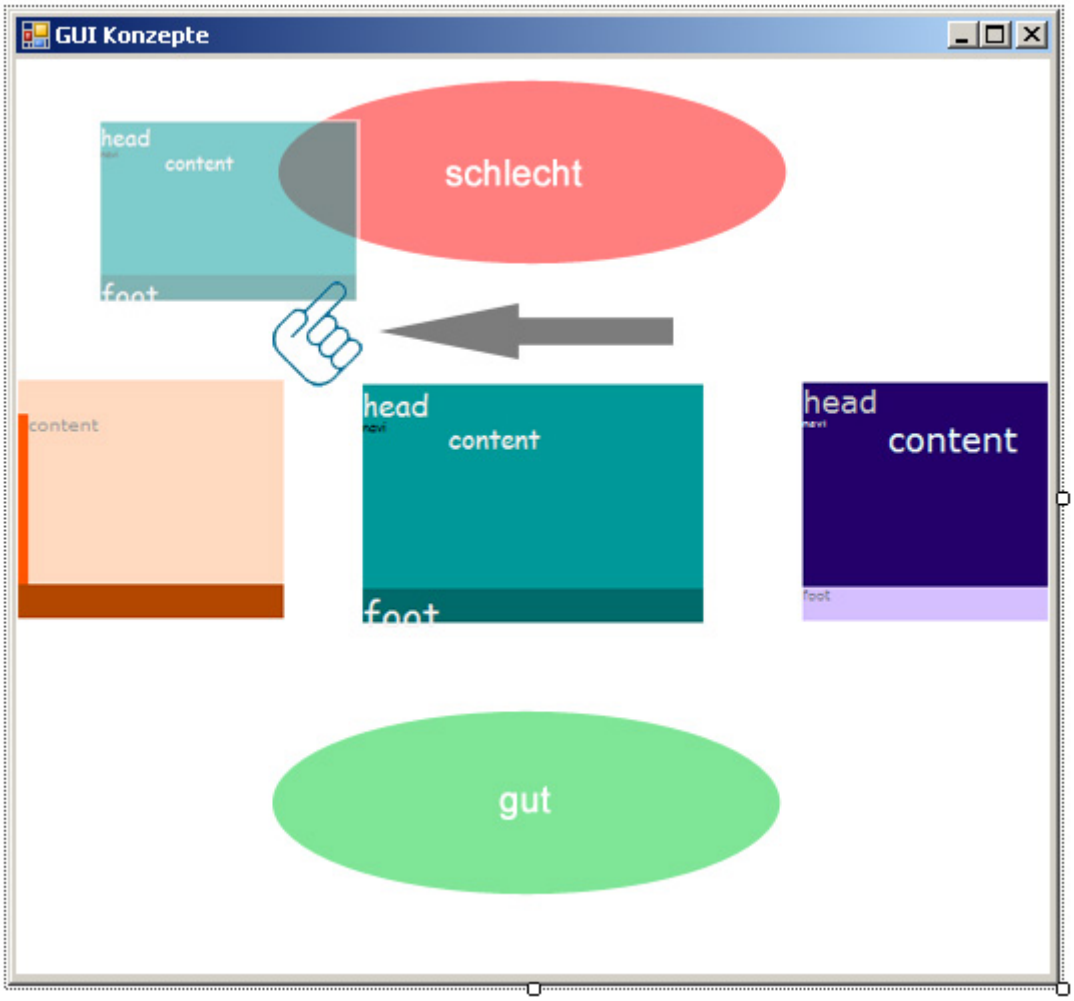

Abbildung 17: Bewertung der Individuen im Drei Klassen Modell

7.4. Das Schauermodell

Die drei bisher vorgestellten Konzepte ähneln sich in Aufbau und Auswertung, dass letzte hier vorzustellende Konzept verfolgt einen anderen Ansatz. Statt eines Laufbandes, dass langsam eine bestimmte Zahl von Individuen auf dem Bildschirm darstellt, wird beim Schauermodell die gesamte, bzw. ein repräsentativer Ausschnitt der gesamten Population dargestellt. Wichtigste Voraussetzung für den Einsatz dieses Modells, ist das ähnliche Individuen auf den Bildschirm in benachbarten Regionen angezeigt werden.

Die dargestellte Abbildung versucht dies zu verdeutlichen. Die einzelnen Stylesheet Individuen werden, abhängig ihrer Parameter, mit anderen ähnlichen Individuen auf dem Bildschirm gruppiert. Der Nutzer hat einen Überblick über den gesamten Populationspool und erhält nun die Möglichkeit bestimmte Bereiche der Population zum Wachsen anzuregen. Dargestellt wird dies durch den gelben Kreis. Durch bewegen dieses "Sonnenscheins" über Bereiche der Population, wird die Selektionswahrscheinlichkeit der beschienenen Individuen erhöht. Die Population verändert sich in Echtzeit, je nachdem welchen Bereich der Benutzer für besonders viel versprechend erachtet.

Abbildung 18: Schauermodell mit geordneter Population und "Sonnenschein" für den Nutzer

8. Fazit

Im Verlauf der Implementierung hat sich gezeigt, dass die zentrale Schwierigkeit in der Entwicklung eines Parametervektors liegt um die CSS-Werte eines Stylesheets sinnvoll zu beschreiben und zu vergleichen. Problematisch ist dabei vor allem die unglaubliche Menge von möglichen CSS-Werten. In der vorliegenden Applikation wurde daher das Hauptaugenmerk auf die Schriftart, Schriftfarbe, Schriftgröße und das Farbschema gelegt. Eine universelle Behandlung beliebiger Stylesheets über eine entsprechende Schnittstelle, wie sie in den Wunschkriterien beschrieben wurde, ist aus derzeitiger Einschätzung, nicht ohne weiteres realisierbar.

Die entwickelte Oberfläche des vorliegenden Programms konnte die Anforderung an einen spielerischen und weitestgehend interaktiven Evaluierungsprozess leider nicht umsetzen. Das Hauptproblem bei der Umsetzung besteht in der Darstellung der Individuen. In der derzeitigen Version wird dazu auf die von der .net-Umgebung zur Verfügung gestellte Klasse *WebBrowser* zurückgegriffen. Das Element stellt alle Funktionalitäten des herkömmlichen Internet Explorers zur Verfügung, dies ist vorteilhaft in der Darstellung führt aber zu Problemen bei der Handhabung dieser Elemente. So bietet dieses Element z.B. nicht die Möglichkeit Mausereignisse wie "onCklick" oder "onMouseOver" abzufragen. Jegliche Mausaktion innerhalb des Browserfensters führen die innerhalb eines Browsers üblichen Aktionen aus, mit der Linken Maustaste kann innerhalb des Elements navigiert werden, die rechte Maustaste ruft das Kontextmenü des Internet Explorers auf. Für die Realisierung der im vorigen Kapitel vorgestellten GUI-Konzepte, ist es notwendig die Anzeigefunktionalität eines Browsers eigenständig zu programmieren. Dies würde es erlauben nur die reine Anzeige von HTML-Dokumenten zu realisieren und Funktionalitäten wie ein Laufband oder Drag&Drop zu implementieren Mit der, durch die .net-Umgebung bereit gestellten Klasse war die Umsetzung solcher Funktionalitäten nicht erfolgreich.

Die hoch gesteckten Ziele bei der Entwicklung einer innovativen grafischen Oberfläche und eines interaktiven und motivierenden Evaluierungsprozesses, konnten leider mit dem vorliegenden Programm nicht im erhofftem Maße erfüllt werden. Auch hat sich herausgestellt, dass moderne Stylesheets mit ihrer ganzen Komplexität die Grenzen der entwickelten Applikation sprengen würden.

Abschließend ist zu sagen, dass sich im Verlauf der Arbeit gezeigt hat, dass eine evolutionäre Entwicklung von Stylesheets durchaus sinnvoll sein kann, allerdings sollte dafür die Anzahl der in die Evolution einfließenden Parameter eines Individuums entsprechend eingegrenzt werden. Durchaus vorstellbar wäre es z.B. die Funktionalität der Applikation als kleinen Webservice zu implementieren, um dem Besucher einer Webseite die Möglichkeit zu bieten ein Stylesheet mit optimalen Farbkontrasten zu entwickeln. Für die komplette Entwicklung komplexer Layouts mit unterschiedlichen Bereichen, sinnvoller Anordnung der einzelnen Elemente, einem logischen Aufbau der Seite und einem ästhetischem Erscheinungsbild erscheint der evolutionäre Ansatz, auf Grund der Komplexität, eher nicht vorteilhaft.

Literaturverzeichnis

[BHS07] Boersch, Ingo; Heinsohn, Jochen; Socher, Rolf: Wissensverarbeitung 2. Auflage Elsevier, Februar 2007

[EV08] http://www.evofit.co.uk/ Stand: 27.07.2008

[Le00] Lewis, M.: Aesthetic Evolutionary Design with Data Flow Networks, 2000 Advanced Computing Center for the Arts of Design, Ohio State University

[Lib07] Liberty, J.: Programmieren mit C#, O'Reilly Verlag 2007 2. Auflage

[Me06] Meyer, E.: Cascading Stylesheets das umfassende Handbuch, O'Reilly Verlag 2007 2. Auflage

[Mu05] Münz, S.: Professionelle Websites, Addison-Wesley Verlag 2005

[Po00] Pohlheim, H.: Evolutionäre Algorithmen, Springer Verlag Berlin Heidelberg,2000.

[Sch05] Schmitt, C.: CSS Kochbuch, O'Reilly Verlag 2005

[Ta01] Takagi, H.: Interactive Evolutionary Computation, Proceedings of the IEEE, Vol.89, Nr.9 (2001)

[TO00] Takagi, H; Ohsaki M.: IEC-based Hearing Aid Fitting. In: IEEE System, Man and Cybernetics (1999), Oktober, Tokyo Japan vol. III, Seite 657-662

[We02] Weicker, K.: Evolutionäre Algorithmen, Taubner Verlag 2002

Abbildungsverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Brandenburg, 18.08.2008 _____